UNIVERSIDAD TÉCNICA DEL NORTE

Facultad de Ingeniería en Ciencias Aplicadas

Escuela de Ingeniería en Sistemas Computacionales

ANÁLISIS Y ESTUDIO DE LA ILUMINACIÓN Y COLOR EN OPENGL

Aplicativo: Desarrollo e Implementación de una Aplicación que simule entornos de trabajo y distribuya de manera eficiente las fuentes de iluminación

Proyecto de Tesis previo la obtención del título de Ingeniero en Sistemas Computacionales

Autor: Brian Daniel Debuire Enríquez

Asesor: Ing. Marcelo Puente Carrera

Ibarra, Octubre 2008

DEDICATORIA

A mi madre, Gulnara Enríquez… con todo mi corazón.

AGRADECIMIENTO

Agradezco primeramente a mi familia y a mi novia Erika, quienes siempre me han dado su apoyo incondicional, gracias a ellos he podido cumplir con mis metas y superar muchos obstáculos.

Mi gratitud al Ing. Marcelo Puente Carrera, quien a lo largo de la realización de este trabajo supo guiarme y motivarme, nunca dudó en compartir su preciado tiempo, experiencia y amplios conocimientos sobre el tema.

También reconozco la excelente labor que ha realizado la Universidad Técnica del Norte (docentes, personal administrativo y trabajadores), al abrirme las puertas hacia el conocimiento a través de la investigación, incentivarme a retribuir hacia la comunidad.

A mis compañeros y amigos de la facultad, con quienes disfruté muchos momentos de alegría y compartí muchas horas de duro trabajo para realizar las distintas tareas y trabajos académicos, de manera que aprendí a trabajar en equipo, a compartir mis conocimientos y experiencias, y ayudar a quien lo necesite.

Agradezco también a Dios, quien me ha dado la fortaleza necesaria para enfrentar muchos inconvenientes y siempre estuvo a mi lado para guiar mi camino.

Finalmente, mil disculpas si estas cortas frases de agradecimiento no sean suficientes para las personas involucradas que aquí hago referencia, y muchas más que talvez esté dejando de lado, espero entiendan que con todo mi corazón por siempre estaré agradecido. GRACIAS.

Brian Debuire Enríquez

RESUMEN

En este trabajo de tesis se pretende estudiar el Modelo de Iluminación de OpenGL para su posterior aplicación en la simulación de ambientes de trabajo iluminados gracias a la distribución de luminarias que se obtiene por el Método de las Cavidades Zonales. De manera que, las lineas de conocimiento sobre la Iluminación y el Color y los Gráficos Computarizados Tridimensionales se unen para dar como resultado un aplicativo que ayude en la tarea de hacer cumplir con los Niveles de Iluminación recomendados por las Normas Nacionales e Internacionales y combatir el Calentamiento Global al hacer un uso eficiente de la energía electrica.

ZonalCavs es el producto de este trabajo, escrito en C++, su interfaz gráfica en Qt, y el encargado del renderizado 3D es OpenGL.

ABSTRACT

This thesis is oriented to the study of the OpenGL Lighting Model for its latter application in the simulation of lit work environments through de distribution of luminaries obtained with the Zonal Cavities Method. This way, Lighting and Color Theory and Three-dimensional Computer Graphics gets together to achieve an application that will help to respect the Illumination Levels recommended by national and international norms, and fight against Global Warming when making an efficient use of the electric power.

ZonalCavs is the product of this work, written in C++ with graphical user interface made in Qt and using OpenGL as the 3D renderer.

CONTENIDOS

CAPÍTULO I 1
1 ILUMINACIÓN Y COLOR 2
1.1 Introducción 2
1.2 Tipos de Luz 2
1.2.1 El Espectro Electromagnético 3
1.2.2 Luz Natural 5
1.2.3 Luz Artificial 5
1.3 Magnitudes y Unidades Radiométricas 6
1.3.1 La energía radiante 6
1.3.2 Potencia radiante o flujo radiante 6
1.3.3 La irradiancia 7
1.3.4 La exitancia radiante 8
1.3.5 El ángulo sólido 8
1.3.6 La intensidad radiante 9
1.3.7 La radiancia 10
1.3.8 Radiancia de una fuente en función de la intensidad radiante 11
1.4 Magnitudes y Unidades Fotométricas 12
1.4.1 Introducción 12
1.4.2 Flujo o Potencia Luminosa 13
1.4.3 Iluminancia 14
1.4.4 Intensidad luminosa 15
1.4.5 Luminancia 16
1.4.6 Eficacia luminosa de una fuente de luz 17
1.5 Confort Visual 19
1.5.1 Introducción 19
1.5.2 Variación temporal de la iluminación 21
1.5.3 Deslumbramiento 21
1.5.4 Sombras 23
1.5.5 Reflexiones de Velo 23
1.6 Alumbrado de Puestos de Trabajo con Pantallas De Visualización de Datos 24
1.6.1 Introducción 24
1.6.2 Recomendaciones y Soluciones 25
1.7 Color y Colorimetría 27
1.7.1 Introducción 27
1.7.2 La Colorimetría Tricromática 28
1.7.3 Campo Físico 31
1.7.4 Campo Fisiológico 40
1.8 Enfermedades Visuales 64
1.8.1 Introduccion 64
1.8.2 Asthenopia 64
1.8.3 Hipermetropía 65
1.8.4 Miopía 66
1.8.5 Presbicia 66
1.8.6 Glaucoma 67
1.9 Fuentes Luminosas 69
1.9.1 Introduccion 69
1.9.2 Gráficos y Diagramas 69

1.9.3 Clasificación ... 73
1.9.4 Características Generales ... 74
1.9.5 Criterios de Selección ... 79
1.10 Métodos de Iluminación ... 81
1.10.1 Introducción ... 81
1.10.2 Alumbrado General ... 81
1.10.3 Alumbrado Localizado ... 82
1.10.4 Alumbrado General Localizado ... 83
1.10.5 Alumbrado Modularizado ... 83
1.11 Métodos de Cálculo de Iluminación de Interiores ... 85
1.11.1 Método de las Cavidades Zonales ... 85

CAPITULO II ... 98
2 GRÁFICOS TRIDIMENSIONALES CON OPENGL ... 99
2.1 Una breve historia sobre los Gráficos en Computadora ... 99
2.1.1 Llega el CRT ... 99
2.1.2 Gráficos en 3D ... 101
2.1.3 2D + Perspectiva = 3D ... 102
2.2 Artificios 3D ... 105
2.2.1 Introduccion ... 105
2.2.2 Perspectiva ... 105
2.2.3 Color y Sombreado ... 106
2.2.4 Luz y Sombras ... 107
2.2.5 Mapeado de Texturas ... 108
2.2.6 Niebla ... 108
2.2.7 Blending y Transparencia ... 109
2.2.8 Antialiasing ... 110
2.3 Usos comunes para los Gráficos 3D ... 111
2.3.1 3D en Tiempo Real ... 111
2.3.2 3D en Tiempo no Real ... 114
2.4 Principios Básicos de la Programación 3D ... 115
2.4.1 Modo Inmediato y Modo Retenido ... 115
2.4.2 Sistemas de Coordenadas ... 116
2.4.3 Proyecciones ... 120
2.5 Introducción a OpenGL ... 124
2.5.1 ¿Qué es OpenGL? ... 124
2.5.2 Breve historia de OpenGL ... 126
2.5.3 Estructura Básica de una Aplicación OpenGL ... 128
2.5.4 Tipos de Datos ... 129
2.5.5 Conveniencias del Nombre de las Funciones ... 130
2.5.6 Librerías y Cabeceras ... 131
2.5.7 El Renderizado en OpenGL ... 132
2.6 OpenGL como una Máquina de Estados ... 137
2.6.1 Generalidades del Dibujo en Computador ... 137
2.6.2 Los Sistemas de Coordenadas ... 141
2.6.3 Dibujado de Primitivas Geométricas en OpenGL ... 143
2.6.4 Manejo Básico de los Estados ... 145
2.6.5 Vectores Normales ... 147
2.7 Vistas ... 149
2.7.1 Trasformaciones de Vista y de Modelos ... 151

2.7.2 Transformaciones de Proyección 154
2.7.3 Transformación del Puerto de Vista 156
2.7.4 Las Pilas de Matrices 157
2.8 Color 161
2.8.1 Generación de Fotones en el Computador 161
2.8.2 Color en OpenGL 162
2.9 Iluminación en OpenGL 167
2.9.1 El modelo de Iluminación de OpenGL 167
2.9.2 Materiales en OpenGL 170
2.9.3 Selección de un Modelo de Iluminación 176
2.9.4 Definición de las Propiedades de los Materiales 179
2.10 Blending, Antialiasing y Fog 183
2.10.1 Blending 183
2.10.2 Antialiasing 187
2.10.3 Fog 190
2.11 Mapas de Bits, Fuentes e Imágenes 192
2.11.1 Mapas de Bits y Fuentes 192
2.11.2 Imágenes 194
2.12 Mapeado de Texturas 198
2.12.1 Introducción 198
2.12.2 Procedimiento 198
2.12.3 Mipmapping 206
2.12.4 Objetos de Textura 208
2.12.5 Texturas Residentes 209
2.13 GLUT: OpenGL Utility Library 210
2.13.1 Introducción 210
2.13.2 Inicialización y creación de ventanas 210
2.13.3 Manejo de eventos de entrada y de la ventana 211
2.13.4 Inicialización y renderizado de objetos tridimensionales 212
2.13.5 Manejo de procesos de fondo 213
2.13.6 Ejecución del Programa 213
2.14 OpenGL vs Direct3D 214
2.14.1 Ventajas de Direct3D 214
2.14.2 Desventajas de Direct3D 215
2.14.3 Ventajas de OpenGL 215
2.14.4 Desventajas de OpenGL 216

CAPITULO III 218
3 SIMULACIÓN DE ENTORNOS DE TRABAJO Y DISTRIBUCIÓN EFICIENTE DE LAS FUENTES DE ILUMINACIÓN 219
3.1 Introducción 219
3.2 Análisis de Requerimientos 220
3.2.1 Requerimientos del Usuario 220
3.2.2 Requerimientos del Sistema 223
3.3 Diseño del sistema 228
3.3.1 Modelo de Casos de Uso 228
3.3.2 Modelo de Contexto 230
3.3.3 Modelo de Bases de Datos 230
3.3.4 Modelo de Objetos 234
3.3.5 Modelo de Interfaces Gráficas de Usuario 244

3.3.6 Arquitectura del Sistema 249
3.4 Implementación 250
3.4.1 Herramientas y Librerías de Desarrollo 250
3.4.2 Compilación 257
3.5 Instalación 261
3.5.1 Microsoft Windows 261
3.5.2 GNU/Linux 262
3.6 Pruebas 263
3.6.1 Aplicación de ZonalCavs en la Sala de Grados de la Facultad de Ingeniería en Ciencias Aplicadas FICA de la Universidad Técnica del Norte. 263
3.7 Evolución 271

CAPITULO IV 272
4 CONCLUSIONES Y RECOMENDACIONES 273
4.1 Conclusiones 273
4.2 Recomendaciones 274
4.3 Verificación de la hipótesis 276
4.4 Posibles temas de tesis 277

BIBLIOGRAFÍA 278

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Onda electromagnética [WEB02] 3
Ilustración 2: El espectro electromagnético [WEB01] 4
Ilustración 3: Irradiancia 7
Ilustración 4: Exitancia radiante 8
Ilustración 5: El ángulo sólido o espacial 9
Ilustración 6: Intensidad radiante 10
Ilustración 7: Radiancia en función de la intensidad radiante 11
Ilustración 8: Eficiencia luminosa espectral V(λ) para visión fotópica [ELIA02] .12
Ilustración 9: Flujo luminoso 13
Ilustración 10: Iluminancia 14
Ilustración 11: Intensidad luminosa 15
Ilustración 12: Luminancia 16
Ilustración 13: Deslumbramiento directo [WEB08]. 22
Ilustración 14: deslumbramiento por reflexión o indirecto [WEB08]. 23
Ilustración 15: Factores que aumentan el riesgo de fatiga visual en los trabajadores que utilizan PVD [BERT00]. 24
Ilustración 16: Relaciones entre los elementos que conforman la temática del color. 28
Ilustración 17: a) Tono b) Saturación c) Claridad [ELIA02] 30
Ilustración 18: El Sistema de Color de Munsell [WEB11]. 31
Ilustración 19: Distribución espectral del Iluminante A [GUSILU]. 32
Ilustración 20: Distribución espectral del Iluminante D65 [GUSILU]. 32
Ilustración 21: Distribución espectral del Iluminante E [GUSILU]. 33
Ilustración 22: Distribución espectral del Iluminante F2 [GUSILU]. 33
Ilustración 23: Distribución espectral del Iluminante F7 [GUSILU]. 34
Ilustración 24: Ley de la reflexión [WEB05] 36
Ilustración 25: Reflexión especular [WEB05] 36
Ilustración 26: Reflexión difusa [WEB05]. 36
Ilustración 27: Reflexión mixta [WEB05] 37
Ilustración 28: Transmisión regular o especular [GUSTRA]. 38
Ilustración 29: Transmisión difusa [GUSTRA]. 38
Ilustración 30: Reflectancia espectral de una tinta cyan sobre un papel blanco [GUSTRA]. 39
Ilustración 31: Reflectancia espectral de un papel blanco [GUSTRA]. 40
Ilustración 32: Diagrama de un ojo humano [WEB01] 40
Ilustración 33: Cantidades relativas de tres colores necesarias para igualar cualquier color monocromático 43
Ilustración 34: El cubo de color RGB [WEB12] 45
Ilustración 35: Curvas de los valores triestímulos del espectro equienergético del Observador Patrón para Colorimetría C.I.E [DESC99]. 46
Ilustración 36: Representación grafica del plano x + y + z = 1 50
Ilustración 37: Diagrama de cromaticidad C.I.E 1931 [WEB07] 51
Ilustración 38: La constancia de color y la falta de constancia de color [WEB06]. 51
Ilustración 39: El metamerismo [WEB06]. 52
Ilustración 40: Espectro de radiación de un cuerpo negro [WEB01]. 53
Ilustración 41: Ubicación de distintas fuentes luminosas en el espacio de color [ELIA02] 54

Ilustración 42: Curvas de valores triestímulos y diagramas de cromaticidad para los observadores C.I.E 1931 y 1964. 54
Ilustración 43: Elipses de Mac Adam y Brown (1949) reproducidas sobre un diagrama de cromaticidad C.I.E. 1931. Ampliadas 10 veces según G. Wyszecki y W. S. Stiles [WEB01]. 56
Ilustración 44: Diagrama de Cromaticidad Uniforme UCS - CIE 1960, según Wyszecki y Stiles. 57
Ilustración 45: Diagrama de Cromaticidad Uniforme UCS- CIE 1976 [WEB06]. 58
Ilustración 46: Sólido de Color para el Espacio L*a*b* C.I.E. [WEB15] 61
Ilustración 47: Dos estímulos de color S_1 y S_2 representados en el Espacio CIELAB. 63
Ilustración 48: Indicatrix de intensidad luminosa lámpara incandescente (izq.). Curva de distribución luminosa (der.) [DESC99]. 70
Ilustración 49: Curva de distribución luminosa (OSRAM GmbH 41473 DECOSTAR® STUDIO IRC-P 35W) 70
Ilustración 50: Luminarias con infinitos planos de simetría (izq.), con dos planos de simetría (centro) y con un plano de simetría (der.) [WEB08]. 71
Ilustración 51: Diagrama isocandela en proyeccion azimutal [WEB10]. 72
Ilustración 52: Diagrama isolux para la superficie a iluminar [WEB10] 73
Ilustración 53: Clasificación de las fuentes luminosas más importantes [ELIA02] 73
Ilustración 54: Alumbrado general [ELIA02] 81
Ilustración 55: Ejemplos de distribución de luminarias en alumbrado general [ELIA02] 82
Ilustración 56: Alumbrado localizado [ELIA02]. 82
Ilustración 57: Alumbrado general localizado [ELIA02]. 83
Ilustración 58: Alumbrado modularizado [ELIA02] 83
Ilustración 59: División de un local en cavidades. 86
Ilustración 60: El coeficiente de utilización. 88
Ilustración 61: Distribución uniforme típica de luminarias [ELIA02]. 90
Ilustración 62: Separación de luminarias de acuerdo a su apertura del haz de luz [WEB08] 91
Ilustración 63: Esquema de mantenimiento de una instalación de iluminación [ELIA02] 92
Ilustración 64: Superficies de distintos materiales en un local [DESC99] 93
Ilustración 65: Distribución final de las luminarias. 96
Ilustración 66: El videojuego Pong de Atari [WEB01] 100
Ilustración 67: Simulación de Olas en granjas para peces [WEB13] 101
Ilustración 68: Diferencias entre un objeto de 2 dimensiones y un objeto tridimensional. 102
Ilustración 69: Un cubo simple. 102
Ilustración 70: Como se ve en tres dimensiones [WRIG04]. 103
Ilustración 71: Tipos de gafas para ver películas en 3D [WEB16]. 103
Ilustración 72: Un cubo 3D a base de líneas [WRIG04]. 104
Ilustración 73: Un cubo 3D sólido más convincente [WRIG04]. 106
Ilustración 74: Un cubo 3D de un solo color [WRIG04]. 106
Ilustración 75: Un cubo 3D cuyas caras tienen distintos colores [WRIG04]. 107
Ilustración 76: Agregando sombras para incrementar el realismo [WRIG04]. 107

Ilustración 77: La aplicación de texturas agrega más detalle sin geometrías adicionales. 108
Ilustración 78: El efecto de niebla provee la ilusión de espacios abiertos amplios [WRIG04]..... 109
Ilustración 79: Ilusión de reflexión utilizando blending [WRIG04]. 110
Ilustración 80: Aliasing vs Antialiasing..... 110
Ilustración 81: Flight Gear, un popular simulador de vuelo basado en OpenGL. 111
Ilustración 82: El escritorio 3D Beryl para Linux. 112
Ilustración 83: El escritorio 3D Aero de Microsoft Windows Vista 113
Ilustración 84: El entorno Mac OSX. 113
Ilustración 85: Imagen prerenderizada de un personaje de Final Fantasy XIII. 114
Ilustración 86: Sistema de Coordenadas Cartesianas [WRIG04]. 117
Ilustración 87: Regiones de Clipping comunes [WRIG04]. 118
Ilustración 88: Un viewport definido como el doble del area de recorte [WRIG04]..... 119
Ilustración 89: Un viewport que ocupa solo una parte de la ventana del cliente [WRIG04]..... 119
Ilustración 90: Sistema Coordenado Cartesiano de 3 dimensiones [WRIG04]. 120
Ilustración 91: Una imagen 3D proyectada en una superficie 2D..... 121
Ilustración 92: Volumen de la proyeccion ortografica..... 122
Ilustración 93: El volumen (frustum) de una perspectiva de proyección..... 123
Ilustración 94: Formato de las funciones de OpenGL [WRIG04]. 131
Ilustración 95: Orden de las operaciones de OpenGL [WEB01]..... 132
Ilustración 96: Aproximación de superficies [OPEN07]. 143
Ilustración 97: Tipos de primitivas geométricas [OPEN07]. 145
Ilustración 98: Dos vectores definidos por 3 puntos en un plano [WRIG04]... 148
Ilustración 99: Un vector normal como el producto cruz de dos vectores [WRIG04]..... 148
Ilustración 100: La analogia de la camara [OPEN07]. 150
Ilustración 101: Matriz en orden de columna [OPEN07]..... 151
Ilustración 102: Rotar y trasladar (izq.), Trasladar y luego rotar (der.) [OPEN07]. 152
Ilustración 103: Matriz identidad..... 152
Ilustración 104: Ajuste de la cámara con glLookAt() [OPEN07] 153
Ilustración 105: Posición por defecto de la cámara [OPEN07]. 153
Ilustración 106: Proyeccion ortografica (izq.), proyeccion en perspectiva (der.) [WRIG04]..... 154
Ilustración 107: Volumen de proyección en perspectiva con el comando glFrustum() [OPEN07]..... 155
Ilustración 108: Volumen de proyección en perspectiva con el comando gluPerspective() [OPEN07] 155
Ilustración 109: Volumen de la proyección ortográfica [OPEN07]. 156
Ilustración 110: Relación entre volumen de visión y el viewport [OPEN07]. .. 157
Ilustración 111: La pila de matrices [WRIG04]. 158
Ilustración 112: Disección de un monitor de computadora. 161
Ilustración 113: Sección de una pantalla LCD. 162
Ilustración 114: El espacio de color RBG [WRIG04]. 163

Ilustración 115: El cubo de color RGB [WRIG04]. 164
Ilustración 116: Sombreado de una línea desde negro hacia blanco [WRIG04]. 165
Ilustración 117: El mapa de colores del Modo Indexado. 166
Ilustración 118: Un objeto iluminado sólo por luz ambiente [WRIG04]. 168
Ilustración 119: Un objeto iluminado por una fuente de luz difusa [WRIG04]. 169
Ilustración 120: Un objeto iluminado por una fuente de luz especular. 169
Ilustración 121: El parámetro GL_SPOT_CUTOFF de un spotlight [OPEN07] 174
Ilustración 122: Resultado del Ejemplo 19 [WRIG04]. 186
Ilustración 123: Línea normal y línea suavizada con antialiasing (der.) [OPEN07]. 187
Ilustración 124: Las ecuaciones de niebla (fog) en forma gráfica [WRIG04].. 191
Ilustración 125: Un bitmap de la letra F [OPEN07]. 192
Ilustración 126: La ubicacion de glRasterPos*() respecto al bitmap. 193
Ilustración 127: Una escena con colores vs una escena con texturas [WRIG04]. 198
Ilustración 128: Coordenadas de textura [WRIG04]. 201
Ilustración 129: Aplicación de coordenadas para rellenar un cuadrado [WRIG04]. 202
Ilustración 130: Aplicación de coordenadas de manera que se obtenga un triangulo [WRIG04]. 202
Ilustración 131: Geometría Iluminada + Textura = Textura sombreada [WRIG04] 203
Ilustración 132: Filtrado de texturas [OPEN07] 204
Ilustración 133: Modo de envoltura GL_REPEAT [OPEN07] 205
Ilustración 134: Modo de envoltura GL_CLAMP [OPEN07] 206
Ilustración 135: Modo de envoltura GL_REPEAT y GL_CLAMP combinados [OPEN07] 206
Ilustración 136: Mipmaps [OPEN07] 207
Ilustración 138: Diagrama de Casos de Uso de ZonalCavs. 228
Ilustración 139: Modelo de Contexto de ZonalCavs. 230
Ilustración 140: Estructura del Catálogo de Luminarias de ZonalCavs. 231
Ilustración 141: Diagrama de Objetos de ZonalCavs. 234
Ilustración 142: Flujo de ventanas de ZonalCavs 244
Ilustración 143: Ventana principal 245
Ilustración 144: Cuadro de diálogo para abrir un archivo fotométrico EULUMDAT 245
Ilustración 145: Cuadro de diálogo del Catálogo, panel de búsqueda de luminarias. 246
Ilustración 146: Cuadro de diálogo del Catálogo, panel de edición del Catálogo. 246
Ilustración 147: Cuadro de diálogo de configuración de parámetros de OpenGL. 247
Ilustración 148: Cuadro de diálogo de Distribución de Luminarias, panel de distribución 2D. 247
Ilustración 149: Cuadro de diálogo de Distribución de Luminarias, panel de detalles de los resultados. 248
Ilustración 150: Cuadro de diálogo de Distribución de Luminarias, panel de distribución 3D. 248

Ilustración 151: Arquitectura general de ZonalCavs 249
Ilustración 152: Arquitectura de Qt [WEB18] 251
Ilustración 153: Compilación de ZonalCavs mediante el comando MAKE. 252
Ilustración 154: Edición del código fuente de ZonalCavs en Dev C++. 253
Ilustración 155: The Gimp en la edicion de texturas de ZonalCavs. 254
Ilustración 156: Consulta SQL dentro del Catálogo de Luminarias (SQLite3). 255
Ilustración 157: Sala de Grados de la Facultad. 264
Ilustración 158: Disposición actual de luminarias y división en zonas y puntos de medición. 264
Ilustración 159: Vista frontal del local y puntos de medición. 265
Ilustración 160: Fotografía de la luminaria utilizada en la sala de grados [WEB20]. 267
Ilustración 161: Distribución de las luminarias en 2D generada por ZonalCavs. 269
Ilustración 162: Simulación 3D de la sala de grados generada por ZonalCavs. 270

ÍNDICE DE TABLAS

Tabla 1: El espectro electromagnético [WEB01]. 4
Tabla 2: Ejemplos de Niveles de Iluminación más específicos [WEB08] 20
Tabla 3: Rangos funcionales de las capacidades del sistema visual humano [ELIA02]..... 42
Tabla 4: Valores triestímulos del espectro equienergético Observador Patrón Normal C.I.E 1931. 46
Tabla 5: Formato de la Matriz de Intensidades 71
Tabla 6: Distribución de la energía emitida en la radiación de distintas fuentes luminosas [CEI96] 74
Tabla 7: Clasificación CIE de luminarias de acuerdo con la distribución del flujo luminoso hacia los hemisferios inferior y superior [ELIA02]. 75
Tabla 8: Vida nominal y depreciación luminosa para distintos tipos de lámparas [NAR00]. 77
Tabla 9: Características fotométricas, colorimétricas y de duración para las lámparas mas representativas de cada tipo. 78
Tabla 10: Caracteristicas aproximadas de los sistemas de iluminacion [ELIA02] 84
Tabla 11: Criterios de separación entre luminarias [WEB08]. 91
Tabla 12: Valores recomendados por [DESC99] para los factores de mantenimiento y de compensación. 92
Tabla 13: Factores de reflexión ρ y superficies S de diferentes tipos de materiales [DESC99]. 93
Tabla 14: Luminaria Prismawrap de 4 lámparas – Coeficientes de Utilización 96
Tabla 15: Porcentajes de reflectancias efectivas de cavidad de cielorraso o suelo. 96
Tabla 16: Factores de corrección para porcentajes de reflectancia efectiva de cavidad de piso diferentes a 20%..... 97
Tabla 17: Tipos de datos disponibles en OpenGL [OPEN07] 129
Tabla 18: Lista de Buffers que se pueden limpiar con glClear() [OPEN07]. ... 138
Tabla 19: Parámetros permitidos para glBegin() [OPEN07]. 144
Tabla 20: Valores de los componentes de luz de un láser rojo. 170
Tabla 21: Valores por defecto para el primer parámetro de glLight*() [OPEN07] 172
Tabla 22: Valores por defecto del primer parámetro de glLightModel*() [OPEN07] 176
Tabla 23: Valores por defecto para pname del método glMaterial*() [OPEN07] 179
Tabla 24: Factores de mezcla (blending) [ASTL04]. 184
Tabla 25: Operaciones matemáticas de cada ecuación permitida [WRIG04] 186
Tabla 26: Valores permitidos para glHint() [OPEN07] 188
Tabla 27: Ecuaciones de niebla soportadas por OpenGL [OPEN07]..... 190
Tabla 28: Formatos de píxeles necesarios como parámetro para glReadPixels() o glDrawPixels() [ASTL04]. 195
Tabla 29: Tipos de Datos usados por glReadPixels() o glDrawPixels() [WRIG04] 195
Tabla 30: Parámetros para glPixelStore*() [WRIG04]. 196
Tabla 31: Formatos de textura internos más comunes [WRIG04]. 199

Tabla 32: Formatos de texels para glTexImage() [OPEN07]..... 199
Tabla 33: Tipos de Datos para los píxeles [OPEN07]. 200
Tabla 34: Comparación de características entre OPenGL y Direct3D [WEB14] 216
Tabla 35: Cálculo de la iluminancia promedio y uniformidad de iluminación.. 265
Tabla 36: Resultados presentados por ZonalCavs..... 269

CAPÍTULO I

ILUMINACIÓN Y COLOR

1 ILUMINACIÓN Y COLOR

1.1 Introducción

Según estudios realizados, para desarrollar al máximo las actividades en un sitio de trabajo, se requiere entre otros factores de una buena iluminación. Es decir, una iluminación apropiada ayuda a conseguir una producción mayor y mejor al evitar que los trabajadores sufran de posibles pérdidas de la visión, dolores de cabeza, migrañas, jaquecas, etc, desmotivación y accidentes [PUEN01].

El estudio de la luz y el color servirá como la base fundamental para lograr una simulación más realista y así poder desarrollar un aplicativo que nos ayude a combatir el problema del consumo excesivo de energía eléctrica, el calentamiento global e incentivar el cumplimiento de las normas vigentes de higiene y seguridad en el trabajo.

1.2 Tipos de Luz

La luz (del latín *lux, lucis*) es una onda electromagnética, compuesta por partículas energizadas llamadas fotones[1], capaz de ser percibida por el ojo humano y cuya frecuencia o energía determina su color [WEB01]. La luz puede provenir de varias fuentes, por ejemplo la luz del Sol, de una lámpara, una llama, etc.

Diferentes teorías se han ido desarrollando para interpretar la naturaleza de la luz hasta llegar al conocimiento actual. La teoría corpuscular (Newton), la teoría ondulatoria (Huyguens), la teoría electromagnética (Maxwell), la teoría cuántica (Planck) y la teoría unificada (De Broglie y Heisenberg). Todos estos estudios y modelos permitieron detectar que todas las radiaciones son de naturaleza electromagnética y solo difieren entre si por su longitud de onda[2].

[1] El fotón es la partícula elemental responsable de las manifestaciones cuánticas del fenómeno electromagnético.
[2] La longitud de una onda (λ) es la distancia entre dos crestas consecutivas.

Una onda electromagnética (véase Ilustración 1) es la forma de propagación de la radiación electromagnética a través del espacio, y sus aspectos teóricos están relacionados con la solución en forma de onda que admiten las ecuaciones de Maxwell[3] [WEB01].

A diferencia de las ondas mecánicas, las ondas electromagnéticas no necesitan de un medio material para propagarse.

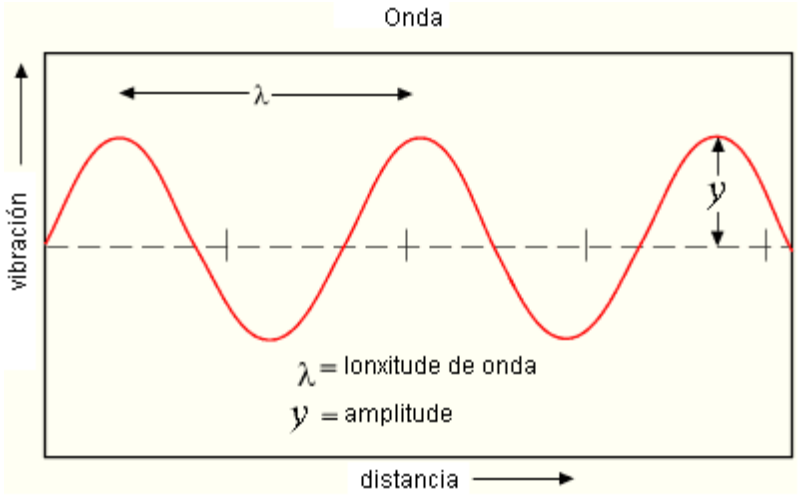


Ilustración 1: Onda electromagnética [WEB02]

1.2.1 El Espectro Electromagnético

El Espectro Electromagnético es la distribución energética del conjunto de las ondas electromagnéticas o, más concretamente, a la radiación electromagnética que emite (espectro de emisión) o absorbe (espectro de absorción) una sustancia. Dicha radiación sirve para identificar la sustancia de manera análoga a una huella dactilar.

El espectro va desde las de menor longitud de onda, como son los rayos cósmicos, los rayos gamma y los rayos X, pasando por la luz ultravioleta, la luz visible y los rayos infrarrojos, hasta las ondas electromagnéticas de mayor

[3] James Clerk Maxwell (1831-1879). Físico escocés conocido principalmente por haber desarrollado un conjunto de ecuaciones que expresan las leyes básicas de la electricidad y magnetismo así como por la estadística de Maxwell-Boltzmann en la teoría cinética de gases.

longitud de onda, como son las ondas de radio. En cualquier caso, cada una de las categorías son de ondas de variación de campo electromagnético [WEB01].

La tabla siguiente muestra el espectro electromagnético, con sus longitudes de onda, frecuencias y energías de fotón:

Tabla 1: El espectro electromagnético [WEB01].

	Longitud de onda (m)	Frecuencia (Hz)	Energía (J[4])
Rayos gamma	< 10 pm	>30.0 EHz	>19.9E-15 J
Rayos X	< 10 nm	>30.0 PHz	>19.9E-18 J
Ultravioleta Extremo	< 200 nm	>1.5 PHz	>993E-21 J
Ultravioleta Cercano	< 380 nm	>789 THz	>523E-21 J
Luz Visible	< 780 nm	>384 THz	>255E-21 J
Infrarrojo Cercano	< 2.5 µm	>120 THz	>79.5E-21 J
Infrarrojo Medio	< 50 µm	>6.00 THz	>3.98E-21 J
Infrarrojo Lejano/submilimétrico	< 1 mm	>300 GHz	>199E-24 J
Microondas	< 30 cm	>1.0 GHz	>1.99e-24 J
Ultra Alta Frecuencia Radio	<1 m	>300 MHz	>1.99e-25 J
Muy Alta Frecuencia Radio	<10 m	>30 MHz	>2.05e-26 J
Onda Corta Radio	<180 m	>1.7 MHz	>1.13e-27 J
Onda Media Radio	<650 m	>650 kHz	>4.31e-28 J
Onda Larga Radio	<10 km	>30 kHz	>1.98e-29 J
Muy Baja Frecuencia Radio	>10 km	<30 kHz	<1.99e-29 J

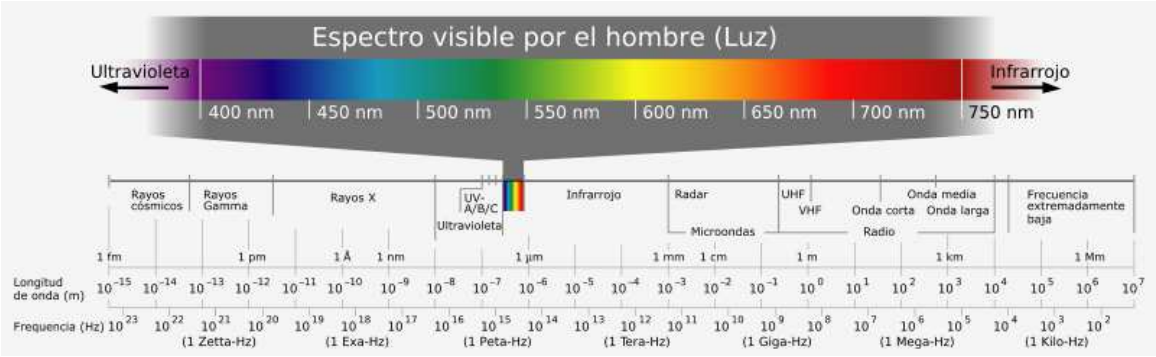


Ilustración 2: El espectro electromagnético [WEB01]

[4] El julio o joule (J) es la unidad del Sistema Internacional para energía y trabajo. Se define como el trabajo realizado por la fuerza de 1 newton en un desplazamiento de 1 metro y toma su nombre en honor al físico James Prescott Joule.

Dentro del punto de vista industrial se puede clasificar a los tipos de luz en: luz natural y luz artificial.

1.2.2 Luz Natural

La luz natural es la que proviene del sol. La cantidad de luminosidad cambia de acuerdo con el tamaño del espacio por donde ingresa al ambiente, y se regula mediante cortinas o equivalentes. Se puede graduar la intensidad de la luz natural que penetra en un ambiente utilizando persianas, cortinas, estores, etc. La tonalidad de la luz natural dependerá de la hora, por las mañanas será blanca y al atardecer rojiza.

1.2.3 Luz Artificial

La luz artificial es indispensable cuando la natural desaparece. Si en una habitación bien decorada no se han tomado en cuenta los cambios de luz, todo su encanto desaparece cuando la iluminación se torna deficiente. Si se conocen y manejan óptimamente los efectos que produce cada tipo de luz artificial, ésta no representará ningún problema.

Este tipo de luz se subdivide en tres tipos:

- **Luz combustible**. Se obtiene del fuego, como las velas, lámparas de petróleo o kerosén, una chimenea, etc. Esta luz es irregular y parpadea mucho, por esto sólo debe utilizarse decorativamente.

- **Luz incandescente**. Despide luz cálida: foco, vela, halógeno.

- **Iluminación de descarga o fluorescente**. Emiten luz blanca y se caracterizan por un bajo consumo de energía: lámparas de fluorescentes.

1.3 Magnitudes y Unidades Radiométricas

1.3.1 La energía radiante

Se puede definir a la luz como la parte de la energía radiante evaluada visualmente, es decir, la energía que al interactuar con alguna superficie, se refleja o se trasmite hacia el sistema visual y produce la respuesta de los fotorreceptores, dotando al ser humano el sentido de la visión [DESC99].

La energía radiante es la que poseen las ondas electromagnéticas como la luz visible, las ondas de radio, los rayos ultravioleta (UV), los rayos infrarrojo (IR), etc. La característica principal de esta energía es que se puede propagar en el vacío, sin necesidad de soporte material alguno. Ej.: La energía que proporciona el Sol y que nos llega a la Tierra en forma de luz y calor [WEB01].

Entonces, la energía radiante Q_e es la energía que se propaga bajo la forma de ondas electromagnéticas y se mide en Joules [J]:

Q_e [J]

1.3.2 Potencia radiante o flujo radiante

Las radiaciones electromagnéticas transportan energía, de forma que un objeto luminoso (radiador) emite energía y cualquier objeto iluminado la recibe. La potencia radiante o flujo radiante P corresponde a un trabajo por unidad de tiempo que es lo mismo que una potencia y por lo tanto su unidad es el Watt [DESC99].

$$P = \frac{J}{s} = watt$$

$$\phi_e = \frac{dQ_e}{dt}[watt]$$

La energía transportada puede manifestarse de formas muy diversas en los cuerpos que la reciben: propiciando reacciones químicas (fotosíntesis y

bronceado), efectos eléctricos (fotocélulas), efectos mecánicos (viento solar), calentamiento (estufas de infrarrojos), etc.

Entonces, definimos al flujo radiante como la energía radiante trasferida en una unidad de tiempo.

Φ_e [w]

1.3.3 La irradiancia

El flujo radiante por unidad de superficie que incide, atraviesa o emerge de un cierto punto A situado sobre una superficie S_0, en cualquiera de las direcciones definidas por el ángulo sólido hemisférico sobre o bajo el punto A es conocido como irradiancia.

$$E_e = \frac{d\Phi_e}{dS_0}[w.m^{-2}]$$

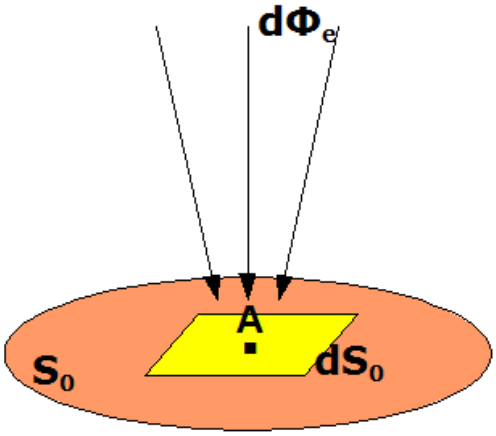


Ilustración 3: Irradiancia

En conclusión, la irradiancia media expresa la densidad del flujo radiante en una superficie receptora S_0 [DESC99].

E_e [w/m^2]

1.3.4 La exitancia radiante

Considerando una superficie emisora A_s, un punto P, y alrededor de ese punto un área elemental dA_s tal como en la Ilustración 4, estamos hablando de un flujo radiante φ_e que abandona la superficie real o imaginaria [DESC99].

$$M_e = \frac{d\phi_e}{dA_s}[w.m^{-2}]$$

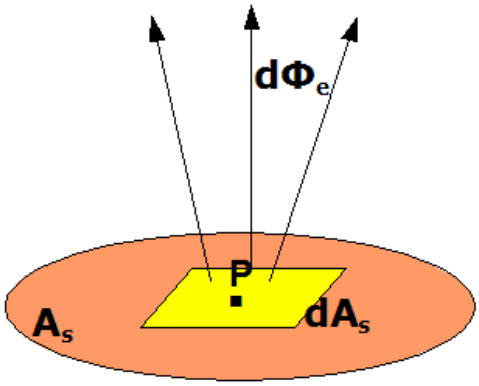


Ilustración 4: Exitancia radiante

Entonces, la exitancia radiante es el cociente entre el flujo radiante que abandona un elemento de superficie que contiene al punto P y el área de ese elemento.

M_e [w/m^2]

1.3.5 El ángulo sólido

El ángulo sólido o ángulo cónico Ω que un objeto abarca, visto desde un punto dado, mide cuan grande aparece ese objeto al observador. La unidad del ángulo sólido es el estereorradián, cuya abreviación es sr. Un estereorradián es igual a un radián[5] al cuadrado [WEB01].

Para calcular el ángulo sólido bajo el cual se ve un objeto desde un punto, se proyecta el objeto sobre una esfera de radio R arbitrario centrada en el punto.

[5] El radián (r) se define como el ángulo que limita un arco de circunferencia cuya longitud es igual al radio de la circunferencia.

Si la superficie de la proyección del objeto sobre la esfera es S el ángulo sólido bajo el cual se ve el objeto es, por definición:

$$\Omega = \frac{S}{R^2}[sr]$$

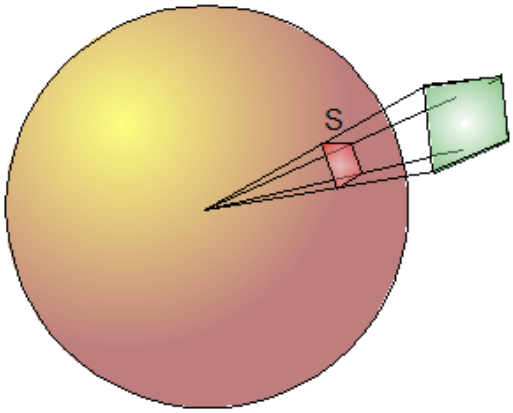


Ilustración 5: El ángulo sólido o espacial

Para precisar la definición de la unidad de ángulo sólido, decimos que estereorradián es el ángulo que teniendo su vértice en el centro de una esfera, determina sobre la superficie de esta un área equivalente a la de un cuadrado cuyo lado es igual al radio de la esfera [DESC99].

Ω [sr]

1.3.6 La intensidad radiante

Se define a la intensidad radiante como el flujo radiante por unidad de ángulo sólido que en cierta dirección r incide, atraviesa o emerge de un cierto punto A situado sobre una superficie S_0.

$$I_e = \frac{d\Phi_e}{d\Omega}[w.sr^{-1}]$$

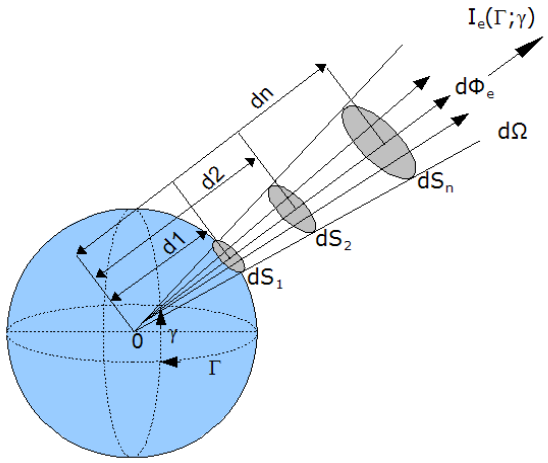


Ilustración 6: Intensidad radiante.

I_e [w/sr]

1.3.7 La radiancia

Esta magnitud resulta muy útil cuando es necesario considerar la radiación de una fuente no puntual o de grandes dimensiones.

La radiancia se define como el cociente entre el flujo radiante que abandona, alcanza o atraviesa un elemento de superficie en ese punto y que se propaga en las direcciones definidas por un cono que contiene la dirección dada, y el producto del ángulo sólido elemental del cono por el área de la proyección ortogonal del elemento de superficie sobre un plano perpendicular a la dirección dada [PUEN01].

L_e [w/sr.m^2]

Su definición resulta compleja por lo que se considerará su definición mas utilizada en luminotecnia conocida como Ley básica de la Radiometría [DESC99].

1.3.8 Radiancia de una fuente en función de la intensidad radiante

Como se vio anteriormente, la radiancia toma en cuenta la dirección en el espacio de la emisión o recepción del flujo radiante. Para las aplicaciones practicas, es importante medir la radiancia en una determinada dirección a partir de la medición de la intensidad radiante en esa misma dirección [DESC99].

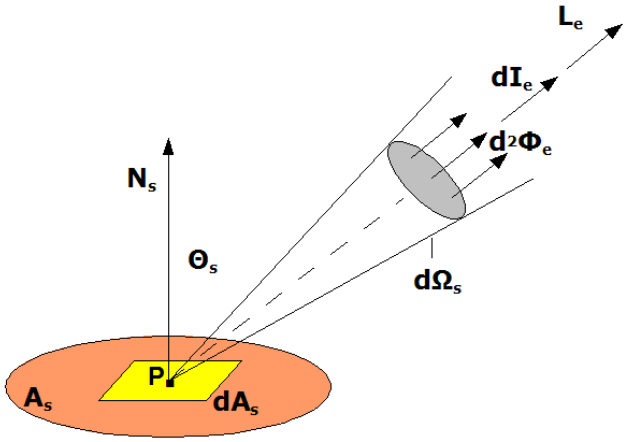


Ilustración 7: Radiancia en función de la intensidad radiante

$$L_e = \frac{dI_e}{dA_s \cos\theta_s}[w.sr^{-1}.m^{-2}]$$

La radiancia media se define como:

$$\overline{L_e} = \frac{I_e}{A_s \cos\theta_s}[w.sr^{-1}.m^{-2}]$$

En caso de que la dirección considerada coincida con l normal N_s la radiancia media resulta:

$$\overline{L_e} = \frac{I_e}{A_s}[w.sr^{-1}.m^{-2}]$$

Es importante mencionar que la radiancia representa una densidad espacial de flujo radiante, a diferencia de la irradiancia que representa una densidad superficial del flujo radiante [DESC99].

1.4 Magnitudes y Unidades Fotométricas

1.4.1 Introducción

El ojo humano no tiene la misma sensibilidad para todas las longitudes de onda que forman el espectro visible (véase El Espectro Electromagnético). La Fotometría[6] introduce este hecho, ponderando las diferentes magnitudes radiométricas medidas para cada longitud de onda por un factor que representa la sensibilidad del ojo para esa longitud.

La función que introduce estos pesos se denomina función de luminosidad espectral o eficiencia luminosa relativa o eficiencia luminosa espectral [DESC99] de un ojo modelo, que se suele denotar como V_λ, V(λ) o $\bar{y}(\lambda)$, siendo λ(lambda) la longitud de onda medida en nanómetros (nm). Esta función es diferente dependiendo de que el ojo se encuentre adaptado a condiciones de buena iluminación (visión fotópica[7]) o de una mala (visión escotópica[8]). Así, en condiciones fotópicas, la curva alcanza su pico para 555 nm, mientras que en condiciones escotópicas lo hace para 507 nm [WEB01].

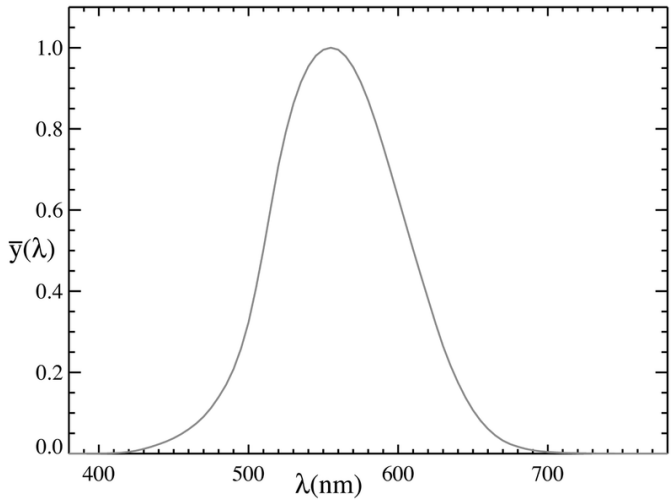


Ilustración 8: Eficiencia luminosa espectral V(λ) para visión fotópica [ELIA02].

[6] La Fotometría es la ciencia que se encarga de la medida de la luz como el brillo percibido por el ojo humano.
[7] La visión fotópica es la visión que tiene lugar con buenas condiciones de iluminación (a plena luz del día). Esta visión posibilita la correcta interpretación del color por el ojo.
[8] La visión escotópica es aquella percepción visual que se produce con niveles muy bajos de iluminación. No es posible una discriminación del color en este tipo de visión: es una visión monocromática.

1.4.2 Flujo o Potencia Luminosa

Se define al flujo o potencia luminosa, como la potencia (W) emitida en forma de radiación luminosa a la que el ojo humano es sensible. La unidad para medir el flujo luminoso es el lumen (lm) y fue determinado empíricamente a partir de la bujía [DESC99]. Siendo el flujo luminoso una potencia, seria lógico utilizar una unidad basada en watts como lo es el watt-luz (w-luz), además gracias a la posibilidad de conversión de magnitudes radiométricas en magnitudes fotométricas.

Con el fin de preservar el lumen, se investigó la relación entre éste y el watt-luz [DESC99]. El proceso no fue sencillo, pero se logró una relación aceptable por la Conferencia Internacional de Pesas y Medidas en 1977 [PUEN01].

1 watt-luz a 555 nm = 683 lm

Entonces, el flujo luminoso queda definido como:

$$\Phi_v = K_m.V(\lambda).\Phi_e(\lambda) \qquad K_m = 683\frac{lumen}{watt}[lm.w^{-1}]$$

Si $\Phi_e(\lambda) = 1watt$ y $V(\lambda = 555nm) = 1$, resulta

$$\Phi_v = 683[lm]$$

Ilustración 9: Flujo luminoso

Φ_v [lm]

1.4.3 Iluminancia

La iluminancia es la magnitud fotométrica que se utiliza para definir el flujo luminoso incidente Φ_v en un punto de una superficie receptora A_r [DESC99]. Su unidad de medida es el lux.

$$1 \text{ lux} = 1 \text{ lumen/m}^2$$

Esta relacionada con el flujo luminoso por la ecuación:

$$\overline{E}_v = \frac{d\Phi_v}{dA_r}[lm.m^{-2}] = lux$$

y con la irradiancia espectral $E_e\lambda$ por la expresión:

$$E_v = K_m \int_{360}^{830} V(\lambda).E_e\lambda d\lambda [lux]$$

En la práctica, al realizar los cálculos de luminotecnia se considera a la superficie receptora como plana, es decir, se calcula una iluminancia media.

E_v [lux] = [lm.m^{-2}]

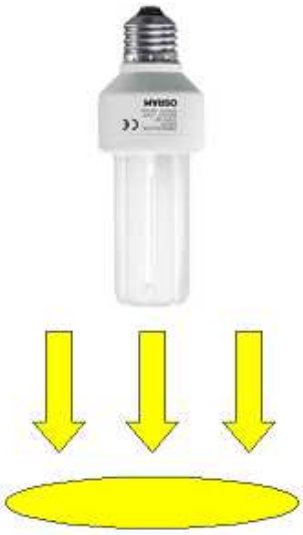


Ilustración 10: Iluminancia

1.4.4 Intensidad luminosa

La intensidad luminosa es la magnitud fotométrica que se utiliza para definir el flujo luminoso emitido por una fuente en una determinada dirección del espacio. Su unidad de medida es la candela (cd), que se define como la intensidad luminosa en una dirección dada, de una fuente que emite una radiación monocromática de frecuencia $540x10^{12}$ Hertz y para la cual la intensidad radiante, en esa dirección, es 1/683 watt por estereorradián [DESC99]. Vale mencionar que todo el conjunto de unidades para las magnitudes fotométricas se deducen de la candela.

La intensidad luminosa esta relacionada con el flujo luminoso por la ecuación:

$$I_v = \frac{d\Phi_v}{d\Omega}[lm.sr^{-1} = cd]$$

y con la intensidad radiante espectral $I_e\lambda$ por la expresión:

$$I_v = K_m \int_{360}^{830} V(\lambda).I_e\lambda d\lambda [cd]$$

I_v [cd]

Ilustración 11: Intensidad luminosa

1.4.5 Luminancia

La luminancia constituye una magnitud apropiada para expresar el valor del estimulo luminoso. Se trata de lo que el ojo realmente ve cuando observa un objeto que emite o refleja una radiación cuyo espectro esta entre 360 y 830 nm. Podemos también decir que la luminancia es la densidad espacial del flujo luminoso [DESC99].

La luminancia esta relacionada con el flujo luminoso por las ecuaciones:

$$L_v = \frac{d^2\Phi_v}{dA_s.\cos\theta_s.d\Omega_s}[cd.m^{-2}] \qquad L_v = \frac{d^2\Phi_v}{dA_r.\cos\theta_r.d\Omega_r}[cd.m^{-2}]$$

y con la radiancia espectral por medio de la ecuación:

$$L_v = K_m \int_{360}^{830} V(\lambda).L_e\lambda d\lambda[cd.m^{-2}]$$

L_v [cd/m^2]

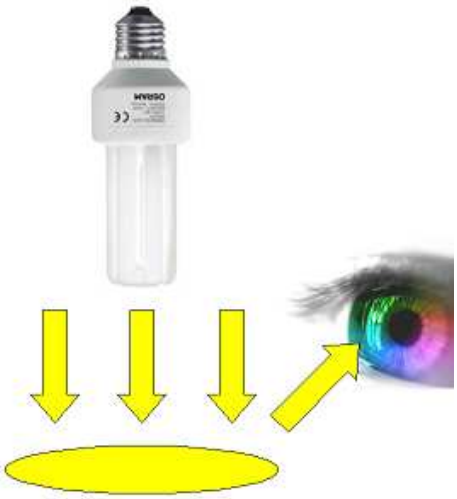


Ilustración 12: Luminancia

1.4.6 Eficacia luminosa de una fuente de luz

La eficacia de una fuente de luz eléctrica se define como el cociente entre el flujo luminoso total emitido y la potencia total consumida por la fuente [DESC99]:

$$\eta = \frac{\Phi_v}{w}[lm.w^{-1}]$$

Este dato es relevante para la elección de una fuente de luz en el momento de iniciar un proyecto de iluminación. Los fabricantes de lámparas proporcionan, entre otras características este dato, de manera que el proyectista obtenga el mayor flujo luminoso en base a un menor consumo de energía, y este es uno de los objetivos principales de esta tesis para poder combatir el problema del calentamiento global.

Los términos eficacia y eficiencia no deben ser confundidos. El primero indica la efectividad con la cual los watt de flujo radiante se convierten en flujo luminoso (K), el segundo término se utiliza para expresar una relación entre dos energías o potencias [DESC99].

Entonces, la eficiencia de una radiación compleja se define como el cociente entre la eficacia luminosa de la radiación K y la máxima eficacia luminosa espectral K_m:

$$V = \frac{K}{K_m} = \frac{\Phi_v / \Phi_e}{K_m} x100[\%]$$

Luego:

- Eficacia luminosa de la radiación $K = \Phi_v / \Phi_e$
- Eficacia luminosa de la fuente $\eta = \Phi_v / W$
- Eficiencia luminosa de la radiación $V = K / K_m$

Ejemplo 1:

Supongamos una lámpara incandescente de 100 w, que trasforma los 100w absorbidos desde la red eléctrica en un flujo radiante total de Φ_e=72 w. Los faltantes 18 w se disipan por varios factores.

Al evaluar el flujo radiante según V(λ) para cada longitud de onda presente y el resultado es multiplicado por K_m, se obtiene un flujo luminoso total de Φ_v=1500 lm.

Entonces calculamos la eficacia luminosa de la radiación:

$$K = \Phi_v / \Phi_e = 1500lm / 72w = 20.8lm.w^{-1}$$

La eficacia luminosa de la fuente:

$$\eta = \Phi_v / W = 1500lm / 100w = 15.0lm.w^{-1}$$

Y la eficiencia luminosa de la radiación:

$$V = K / K_m = 20.8lm.w^{-1} / 683lm.w^{-1} = 3\%$$

Interpretando los resultados se puede apreciar que la eficiencia de la lámpara es de solo un 3% respecto a una fuente ideal. Los fabricantes de lámparas indican la eficacia luminosa de sus productos y no la eficiencia.

Las lámparas de descarga de gas son más eficaces que las incandescentes y la mejor de todas resulta la lámpara de descarga de vapor de sodio de baja presión, la cual obtiene un valor de η=180 $lm.w^{-1}$ [DESC99].

1.5 Confort Visual

1.5.1 Introducción

Las instalaciones de iluminación tienen que estar diseñadas de manera que aseguren un buen rendimiento y confort o comodidad visual. Cada tarea posee un determinado grado de dificultad visual y una alta exigencia visual provoca una perdida de confort visual. Por ejemplo, al trabajar con textos que contienen letras muy pequeñas, la tendencia es acercar el documento hacia los ojos para incrementar el tamaño angular de las letras, lo que implica una fatiga muscular y en consecuencia la reducción del confort visual [ELIA02].

La ergonomía visual, es la encargada de brindar normas que ayudan a mantener la comodidad visual para realizar las actividades de una manera confortable y productiva. Ya que la vista es el principal sentido para la percepción de información, es muy importante cuidarla con una iluminación adecuada y confortable. El nivel de iluminación (véase Tabla 2) de un puesto de trabajo es determinado en función de la tarea a realizar, la edad del trabajador y las condiciones reales en que se debe realizar ciertas tareas.

Es importante citar el trabajo de la **Iniciativa para la Iluminación Eficiente** (ELI) de Argentina [ELIA02], donde se expresa claramente cómo la iluminación influye en la productividad de las personas e incentivan la utilización de nuevos métodos y tecnologías en beneficio del medio ambiente. En contraste, la normativa ecuatoriana es muy vaga respecto a los niveles de iluminación mínimos establecidos, ya que presenta niveles de iluminación muy generales y anticuados.

Tabla 2: Ejemplos de Niveles de Iluminación más específicos [WEB08]

Tareas y clases de local	Flujo Luminoso (lux)		
	Mínimo	**Recomendado**	**Óptimo**
Zonas generales de edificios			
Zonas de circulación, pasillos	50	100	150
Escaleras, escaleras móviles, roperos, lavabos, almacenes y archivos	100	150	200
Centros docentes			
Aulas, laboratorios	300	400	500
Bibliotecas, salas de estudio	300	500	750
Oficinas			
Oficinas normales, mecanografiado, salas de proceso de datos, salas de conferencias	450	500	750
Grandes oficinas, salas de delineación, CAD/CAM/CAE	500	750	1000
Comercios			
Comercio tradicional	300	500	750
Grandes superficies, supermercados, salones de muestras	500	750	1000
Industria (en general)			
Trabajos con requerimientos visuales limitados	200	300	500
Trabajos con requerimientos visuales normales	500	750	1000
Trabajos con requerimientos visuales especiales	1000	1500	2000
Viviendas			
Dormitorios	100	150	200
Cuartos de aseo	100	150	200
Cuartos de estar	200	300	500
Cocinas	100	150	200
Cuartos de trabajo o estudio	300	500	750

Existen muchos casos o situaciones donde se perjudica el confort visual y que se describen a continuación.

1.5.2 Variación temporal de la iluminación

La variación temporal de la iluminación, también conocida como parpadeo, es un fenómeno común en tubos fluorescentes, pantallas de televisión o monitores de computadora, luces estroboscópicas decorativas, etc. Este parpadeo es nocivo para la salud, sobre todo en frecuencias muy bajas y en personas jóvenes.

Se han registrado quejas frecuentes de dolores de cabeza en trabajadores de oficinas con iluminación fluorescente, con y sin pantallas de computadoras. Cuando se introducen balastos[9] electrónicos, se eleva la frecuencia del parpadeo, lo que disminuye los dolores de cabeza y aumenta el bienestar e incluso disminuye la tensión nerviosa de las personas más sensibles [ELIA02].

1.5.3 Deslumbramiento

El deslumbramiento es definido por [DESC99] como "*toda reducción mas o menos persistente de la capacidad de percepción visual o toda molestia resultante de la presencia en el campo visual de objetos muy brillantes o de la visión simultanea de luminancias muy diferentes*." En pocas palabras, el deslumbramiento es una sensación molesta que se produce cuando la luminancia de un objeto, es mucho mayor que la de su entorno. Por ejemplo al mirar directamente una bombilla eléctrica o el Sol.

Existen varios tipos de deslumbramiento:

1.5.3.1 Deslumbramiento fisiológico

Es la reducción de las funciones básicas de la visión. Este tipo de deslumbramiento es muy común en las calles durante la noche por causa de los automóviles que circulan en sentido opuesto. En muchos casos, este tipo de deslumbramiento no afecta el confort, por ejemplo la libre elección de una persona de pasar muchas horas frente a un televisor o computador [ELIA02].

[9] El balasto electrónico se compone de un circuito rectificador diodo de onda completa y un oscilador, encargado de elevar la frecuencia de la corriente de trabajo de una lámpara fluorescente.

1.5.3.2 Deslumbramiento psicológico

Este tipo de deslumbramiento no produce ningún cambio en el rendimiento visual, pero si causa disminución de confort debido a fuentes luminosas en el campo periférico que producen distracción de la tarea que se esta realizando.

Se debe emplear las recomendaciones locales o curvas de luminancias límites del Método de Söllner adoptadas por las normas IRAM-ADDL (IRAM –J20) para estimar la magnitud del deslumbramiento psicológico. El uso de superficies de alta reflectancia en el ambiente, como el aumento de la luminancia de fondo contra la que son vistas las luminarias, ayuda a reducir este tipo de deslumbramiento [ELIA02].

1.5.3.3 Deslumbramiento por niveles de luz excesivos

Este deslumbramiento es causado por niveles de luz excesivos en la escena visual, que enceguece y produce fotofobia. Por ejemplo al leer bajo la luz del sol, el observador entorna sus ojos, parpadea o mira lejos, produciendo una situación incomoda en la tarea que esta realizando. Las soluciones en este caso pasan por el apantallamiento de la radiación [ELIA02].

El deslumbramiento puede producirse por dos formas:

1.5.3.4 Deslumbramiento directo

El deslumbramiento directo es el producido por las luminarias o fuentes de luz visibles que aparecen en el campo visual del observador.

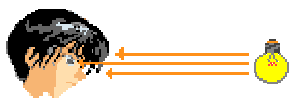


Ilustración 13: Deslumbramiento directo [WEB08].

1.5.3.5 Deslumbramiento por reflexión

El deslumbramiento por reflexión o indirecto es producido por las reflexiones del lugar y/o la superficie de trabajo (mesas, muebles, cristales, espejos, etc).

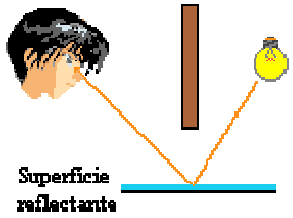


Ilustración 14: deslumbramiento por reflexión o indirecto [WEB08].

1.5.4 Sombras

Las sombras pueden constituir un grave problema al dejar en la oscuridad zonas de interés dentro de una escena. El tamaño de una sombra depende del tamaño del objeto, la puntualidad y proximidad de la fuente de luz hacia el objeto [ELIA02]. Las sombras pueden provocar molestias visuales al exigir adaptación en tiempos cortos, auque también las sombras ayudan a percibir de mejor manera los objetos tridimensionales.

Un ejemplo cotidiano de este problema se encuentra en las personas que trabajan en mesas de dibujo, donde la sombra de la mano perjudica en ciertas ocasiones la completa visibilidad del trabajo. Entonces, este problema se lo puede minimizar al colocar fuentes de luz adicionales o correctamente colocadas.

1.5.5 Reflexiones de Velo

La reflexión o luminancia de velo es aquella que deriva de la mala ubicación de las fuentes de luz con respecto al plano de trabajo, produciendo un intenso brillo sobre la superficie de lectura que vela parcial o totalmente el texto haciéndolo prácticamente ilegible. Se deberá modificar la posición de las luminarias. La iluminación debe provenir de los laterales y nunca de arriba [WEB09].

1.6 Alumbrado de Puestos de Trabajo con Pantallas De Visualización de Datos

1.6.1 Introducción

En las empresas, la mayoría de los trabajadores interactúan con pantallas de video o de visualización de datos (PVD), y en muchos casos, el principal componente de la tarea visual del trabajador es la pantalla. Este tipo de entorno genera tensión en la actividad visual. Un buen diseño de la iluminación en oficinas requiere tomar en cuenta muchos factores relacionados con el uso de monitores o pantallas para poder reducir la fatiga visual (véase Ilustración 15) y mejorar el confort del trabajador y por consecuencia el desempeño de sus tareas.

Ilustración 15: Factores que aumentan el riesgo de fatiga visual en los trabajadores que utilizan PVD [BERT00].

Como se presento anteriormente, hay varios factores que perjudican el confort visual, como por ejemplo el deslumbramiento. A continuación se presentan varias soluciones sugeridas por Luis Deschères [DESC99] para el correcto alumbrado de puestos de trabajo que involucran en uso de pantallas de video.

1.6.2 Recomendaciones y Soluciones

1. Disponer el puesto de trabajo de modo tal que se eviten los reflejos o luminancias.

2. Utilizar pantallas de fondo claro, los reflejos son menores.

3. Eliminar las superficies de luminancias altas, vistas a través de las ventanas utilizando cortinas opacas.

4. Utilizar pinturas mate en las superficies del local con las siguientes reflectancias:

    a. Cielorraso menor a70%
    b. Paredes entre 40% y 60%
    c. Piso alrededor de 30%

5. Limitar la luminancia de las luminarias en alumbrado directo. La práctica ha demostrado que la luminancia de la luminaria, debe ser fuertemente reducida para γ mayor a 50/60º.

6. Colocar una visera sobre la pantalla. Esta solución es el último recurso [BERT00] ya que es necesario equilibrar bien las luminancias del conjunto y evitar posturas difíciles y penosas.

7. Elegir una superficie de pantalla especialmente tratada para reducción de reflejos (filtros polarizados). Estos procedimientos reducen las reflexiones especulares pero generalmente afectan la nitidez de textos y su luminancia. Se los recomienda cuando todas las otras soluciones no han sido posibles.

Adicionalmente Diane Berthelette [BERT00] recomienda tomar en cuenta factores físicos como:

1. La distancia entre la pantalla y los ojos.
2. El ángulo de lectura, que determina la posición de la cabeza y del cuello.
3. La distancia a las paredes y ventanas.
4. La calidad de los documentos impresos (con frecuencia muy mala);
5. La luminancia de la pantalla y del entorno (para la luz natural y artificial);
6. Los efectos de parpadeo;
7. Las fuentes de reflejos y deslumbramientos,
8. El nivel de humedad.

En conclusión, el cumplimiento de estas recomendaciones además de una correcta iluminación, garantizará la salud y el confort de los trabajadores.

1.7 Color y Colorimetría

1.7.1 Introducción

La necesidad de entender el color es notable para proyectistas en iluminación, ingenieros, arquitectos, diseñadores o decoradores de interiores, especialistas en las fábricas textiles, tintoreros, etc. Específicamente, en la industria textil, la necesidad de contar con una buena reproducción del color es indispensable, por ejemplo, en los procesos de tintorería, tal como se manifiesta en [PUEN01].

Generalmente, cuando hablamos de color decimos que es el atributo de un objeto natural o artificial, de una luz, de un espacio o de un medio tal como el cielo o el mar. Según [DESC99], este concepto exige o ha exigido la presencia de tres conjuntos:

- La **fuente** de energía o su manifestación.

- El **objeto** que se halla ubicado en el campo luminoso o, mejor aun, en el campo donde reina esta energía.

- El **observador** que recibe la información energética especialmente por medio de su órgano responsable de la visión, es decir el ojo.

Para el observador, la luz puede provenir directamente de una fuente conocida como primaria, o la luz puede ser producto de reflexiones, difusiones, etc., es decir, se trata de una fuente secundaria de luz.

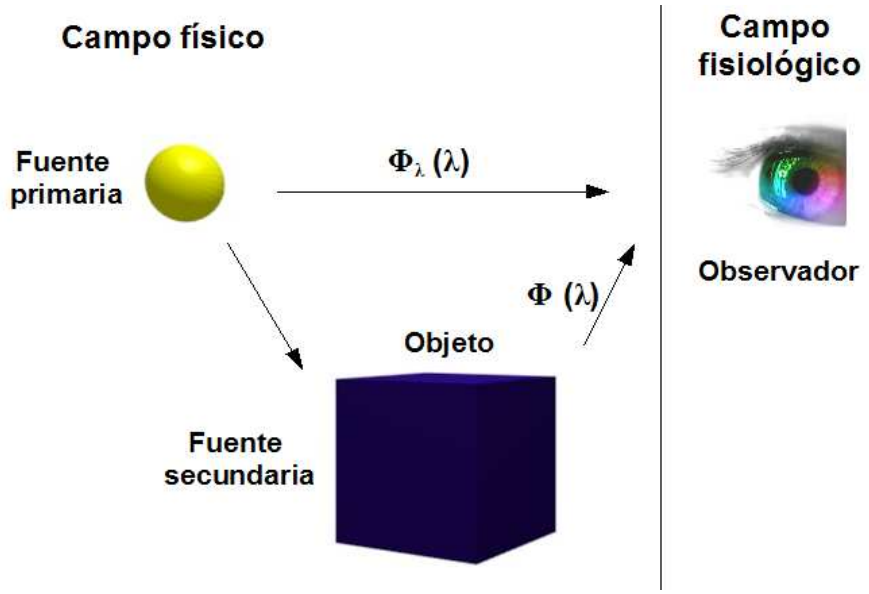


Ilustración 16: Relaciones entre los elementos que conforman la temática del color.

Los diferentes elementos que conforman la temática del color han sido agrupados como se aprecia en la Ilustración 16. Dentro del Campo Físico se encuentran la fuente primaria, la fuente o fuentes secundarias y el objeto, que según sus características físicas, puede reflejar o transmitir la luz que incide en él y así convertirse en una fuente secundaria. La sensación de luz y de color que es la respuesta del observador a un estimulo es en si misma muy compleja. Esta sensación obliga la incorporación de un Campo Fisiológico [DESC99]. Bajo este punto de vista, la radiación física será considerada como un estimulo y se la ubicará en el punto de unión entre el campo físico y el campo fisiológico.

1.7.2 La Colorimetría Tricromática

La colorimetría[10] tricromática, tal como se la conoce actualmente, no tiene muchos años de existencia, aunque los primeros intentos por medir y comprender los conceptos relativos al color se remontan a Aristóteles (384-322 a. C.).

Si tuviésemos que definir el color, quizás una de las más simples y directas formas sería la dada por JUDD [DESC99] quien expresa que:

[10] La colorimetría es la ciencia que estudia la medida de los colores y que desarrolla métodos para la cuantificación del color, es decir, para la obtención de valores numéricos del color.

“si dos objetos de igual forma y textura, iluminados con la misma luz y en iguales condiciones de observación pueden diferenciarse, el atributo de estos objetos que produce esa diferenciación es el color”

Si se desea una definición más rigurosa podría decirse:

“color es el atributo de la luz que hace corresponder unívocamente a cada distribución espectral una sensación. Esta sensación esta condicionada por la intensidad y duración del estimulo, el estado de adaptación del observador, el área de la retina afectada y el contraste luminoso y cromático con que se percibe”

El color depende del observador que integra el campo fisiológico donde se produce la sensación de color que es un atributo psicológico. Además, y según como se afirma en [DESC99], el fenómeno del color en su concepción integral es psicofísico y queda especificado por los valores triestímulos (véase 1.7.4.2 Trivariancia Visual) de la radiación que penetra en el ojo.

La colorimetría tricromática se basa en la suposición de que existe un sistema trireceptor en la visión. Toda distribución espectral de la radiación que llega a su retina es evaluada según tres parámetros lo que implica un proceso de integración. El observador, cuando ve un color, puede discriminar el tono, claridad y saturación de dicho color. El mundo del color es una mezcla de estos tres atributos.

1.7.2.1 Tono

El tono es un atributo asociado con la longitud de onda dominante en una mezcla de ondas de luz. Así, el tono representa el color percibido por el observador; cuando llamamos a un objeto rojo, naranja o amarillo estamos especificando el tono [WEB04].

1.7.2.2 Claridad

La claridad implica la noción que percibimos de la intensidad de luz en un objeto reflectante, es decir, que refleja la luz pero no tiene luz propia. El intervalo de claridades está comprendido entre el blanco y el negro pasando por todos los grises [WEB04]. Este término se lo utiliza cotidianamente cuando expresamos que un mismo color es más claro o más oscuro.

1.7.2.3 Saturación

La saturación se refiere a que tan puro es el color, es decir, cuánto blanco se mezcla con él. Se parte del color blanco hasta llegar al color totalmente saturado [WEB04].

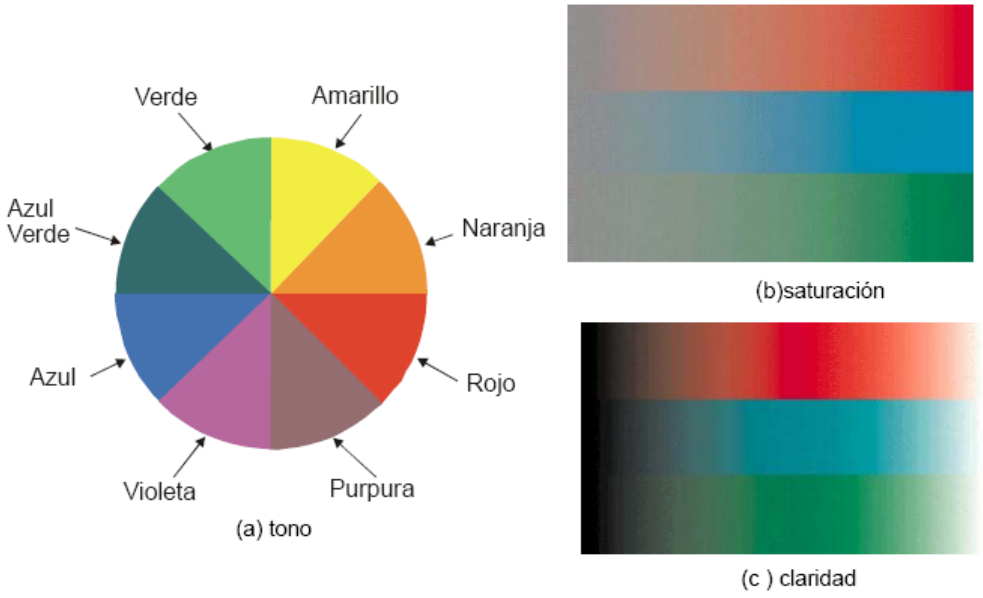


Ilustración 17: a) Tono b) Saturación c) Claridad [ELIA02]

Existen dos maneras de representar el color de la luz: el atlas de color y el sistema calorimétrico de la CIE. Según [ELIA02] el atlas de color mas conocido es el Sistema de Color de Munsell[11] [IESN93], en el cual la posición de cualquier color se identifica con un código alfanumérico que tiene tres términos que indican el tono (hue), claridad (value) y nivel de saturación (chroma).

[11] Albert Henry Munsell, pintor y profesor de arte estadounidense. (1858 – 1918)

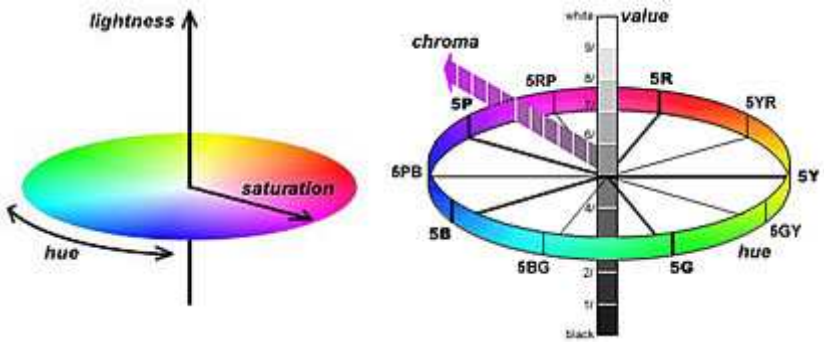


Ilustración 18: El Sistema de Color de Munsell [WEB11].

Los métodos para medir el color y que actualmente se utilizan según [PUEN01] en todo el mundo son los desarrollados por la C. I. E. (Commission Internationale de l'Eclariage). Los dos mas conocidos son:

- El Espacio Cromático Y y X (1931) basado en los valores triestímulos X Y Z definidos por la C. I. E.

- El Espacio Cromático LAB (1976) que provee diferencias de color mas uniformes con referencia a las diferencias visuales.

1.7.3 Campo Físico

1.7.3.1 Estudio de las fuentes primarias

Dentro de las lámparas utilizadas corrientemente en la iluminación, las distribuciones espectrales pueden ser de tipo monocromáticas, de líneas, mixtas o continuas. Cada una de ellas producirá una sensación de color para el ojo. Para uso en colorimetría, la C.I.E ha definido fuentes primarias normalizadas denominadas *iluminantes*.

Iluminante A

El iluminante A se basa en la fuente más usual de luz artificial: La bombilla incandescente de filamento de tungsteno. Su distribución espectral se corresponde con la de un cuerpo negro[12] a unos 2.856 K.

[12] Un cuerpo negro es un objeto teórico o ideal que absorbe toda la luz y toda la energía radiante que incide sobre él.

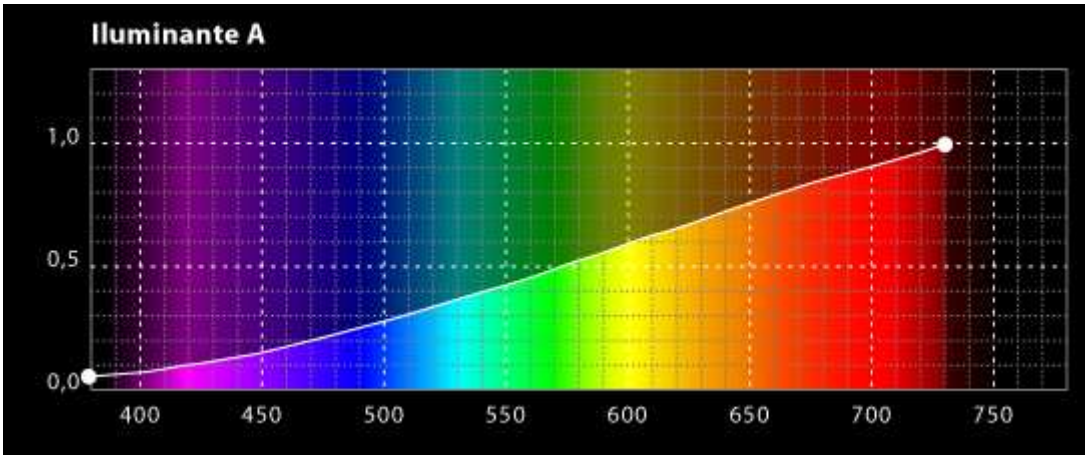


Ilustración 19: Distribución espectral del Iluminante A [GUSILU].

Iluminante C

Representa la luz media del día con una temperatura de color correlacionada alrededor de 6800 K. El iluminante C corresponde a una fuente A con el agregado del un filtro especificado por la C.I.E.

Iluminante D

Los nombres de la serie de luz de día (daylight) comienzan con la letra D mayúscula y dos cifras que indican la temperatura de color aproximada. Así D50 tiene una temperatura de unos 5.000 K. Cuando el sol es de mediodía, su temperatura de color ronda los 5.000 K. Cuando está en el horizonte, su temperatura es inferior. Cuando hay nubes en el cielo, tiene unos 6.500 K, mientras que a la sombra es de 7.500 K.

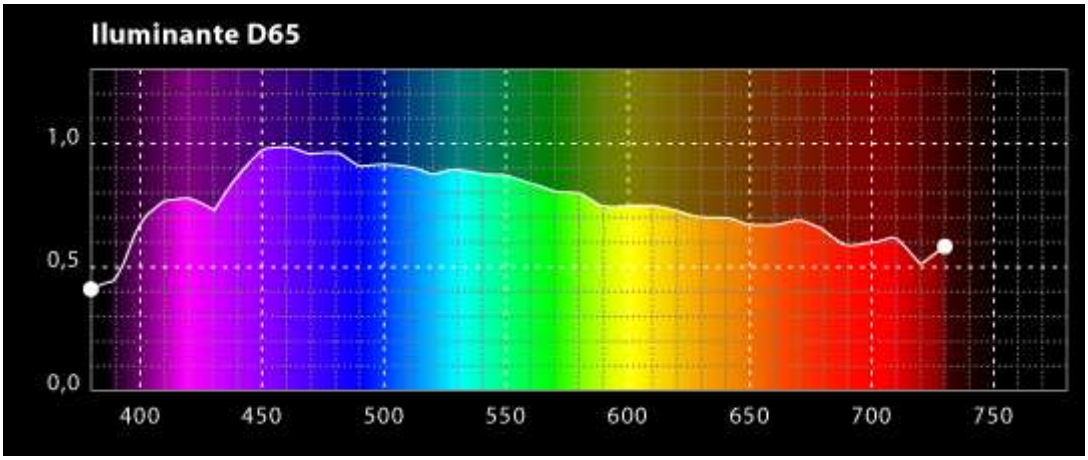


Ilustración 20: Distribución espectral del Iluminante D65 [GUSILU].

Iluminante E

El iluminante estándar E es equienergético (tiene la misma potencia en todas las longitudes de onda del espectro luminoso). Es un iluminante teórico que se usa para cálculos colorimétricos.

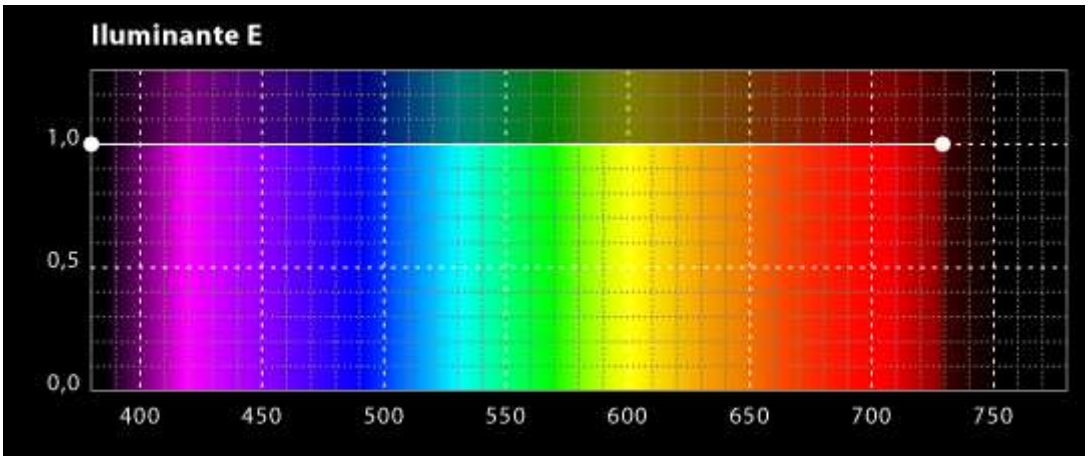


Ilustración 21: Distribución espectral del Iluminante E [GUSILU].

Iluminante F

La C.I.E. indica las distribuciones espectrales de energía en valores absolutos ($\mu W.nm^{-1}.lm^{-1}$) tabuladas entre 380 nm y 780 nm a intervalos de 5 nm para 12 F-iluminantes que representan 12 diferentes tipos de lámparas fluorescentes. En particular se recomienda que los iluminantes F2, F7 y F11 son prioritarios sobre el resto cuando se trata de seleccionar algunos iluminantes típicos.

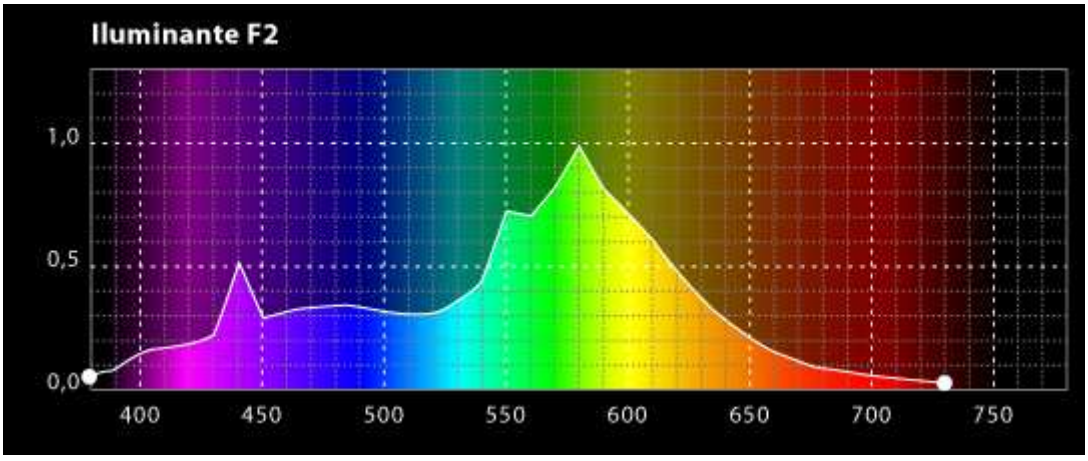


Ilustración 22: Distribución espectral del Iluminante F2 [GUSILU].

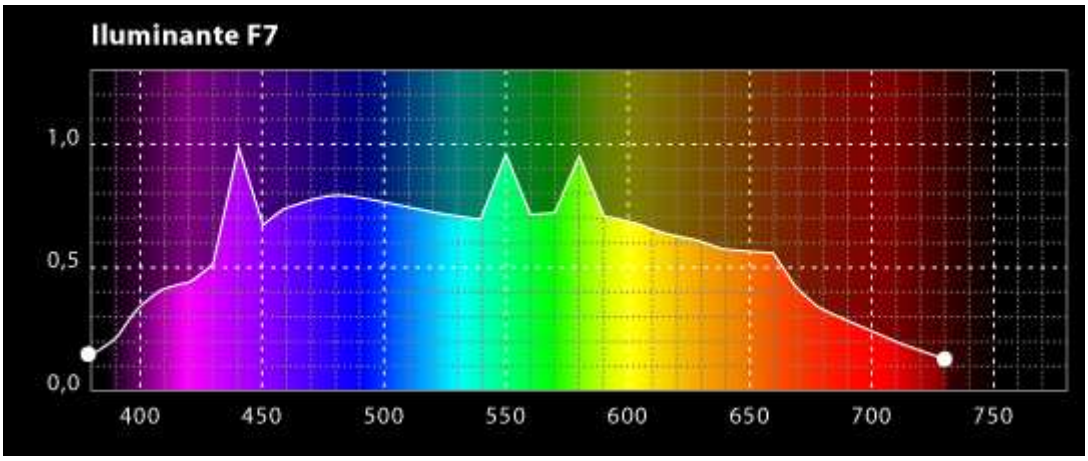


Ilustración 23: Distribución espectral del Iluminante F7 [GUSILU].

Los datos espectrales de los iluminantes estándares CIE (tabulados en intervalos de 1 y 5 nanómetros) se hallan en el sitio web del **Laboratorio Munsell de la Ciencia del Color** (http://www.cis.rit.edu/mcsl/) en forma de hojas de cálculo. Los del iluminante A y D65 se hallan también en el sitio web **Color & Vision Database** (http://www.cvrl.org/).

1.7.3.2 Estudio de las fuentes secundarias

Como se puede apreciar en la Ilustración 16, la fuente secundaria u objeto recibe la energía radiante de la fuente primaria. Esa energía puede ser absorbida, transmitida, reflejada y/o difundida. Si la fuente es una luz cuya distribución espectral esta compuesta por la suma de todas las radiaciones del espectro en proporciones aproximadamente iguales, da lugar a una sensación luminosa cuyo color es prácticamente nulo [DESC99]. Se trata de luz blanca también llamada *acromática*.

Esta luz acromática al incidir sobre el objeto, será la responsable de que veamos el objeto en colores. Si toda la luz acromática es reflejada, el objeto se verá blanco. Si toda es absorbida se verá negro o absente de color. Y si es parcialmente absorbida y parcialmente reflejada, se vera coloreado de acuerdo a las longitudes de onda reflejadas.

Si la luz incidente es monocromática[13], con la luz blanca el objeto se verá blanco ya que refleja todas las longitudes de onda. Con la luz amarilla se verá amarillo, ya que es la única longitud de onda que existe para reflejar.

Comúnmente utilizamos el color para señalar una propiedad del objeto. Pero esto no es cierto, el objeto intrínsicamente no posee color alguno. El objeto solo puede reflejar la luz que incide en él. Y el color del cuerpo dependerá de la distribución espectral de la luz que lo ilumina [DESC99].

Difusión

Las radiaciones están sometidas a un cierto número de reflexiones aleatorias en la materia y se esparcen en todas direcciones. Este fenómeno explica el color azul del cielo. La difusión esta representada por la ley:

$$D = \frac{C^{te}}{\lambda^4}$$

En iluminación hablamos de reflectancia y transmitancia y las definimos para una radiación incidente de distribución espectral, polarización y modo de irradiación dadas.

Reflectancia

Es la relación entre el flujo luminoso reflejado y el incidente. Expresado en forma porcentual, resulta:

$$\rho(\%) = \frac{\Phi_\rho}{\Phi}.100$$

Por ejemplo, para un flujo luminoso incidente (Φ) de 3200 lm, y un flujo luminoso reflejado (Φ_ρ) de 2500 lm, la reflectancia será de 78%.

[13] A diferencia de la luz blanca, que está formada por muchos componentes, la luz monocromática es aquella que está formada por componentes de un solo color. Es decir, que tiene una sola longitud de onda, correspondiente al color.

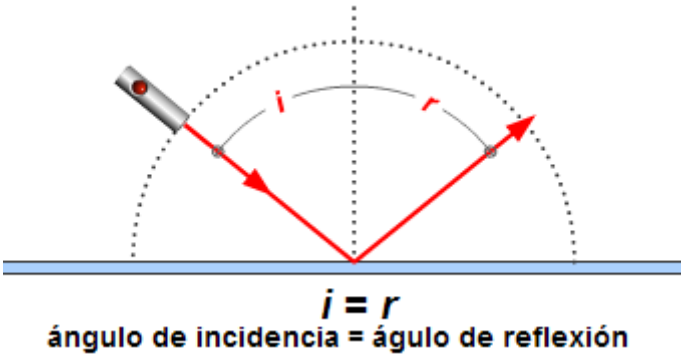


Ilustración 24: Ley de la reflexión [WEB05]

En general, la reflexión sobre una superficie es mixta y esto significa que está compuesta por:

- **La reflexión regular o especular**: ρ_r que es una reflexión sin difusión y que obedece a las leyes ópticas validas para los espejos.

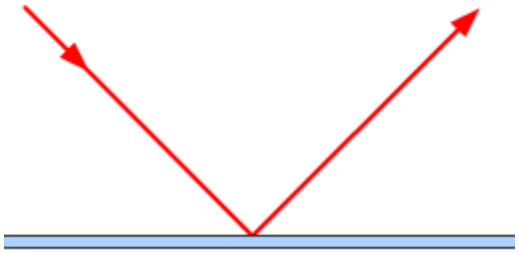


Ilustración 25: Reflexión especular [WEB05]

- **La reflexión difusa:** ρ_d en la cual no se manifiesta la reflexión especular.

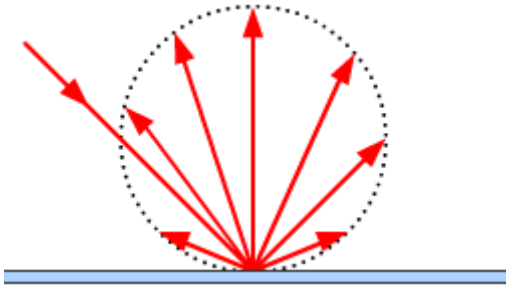


Ilustración 26: Reflexión difusa [WEB05].

Para la reflexión mixta tenemos: $\rho = \rho_r + \rho_d$

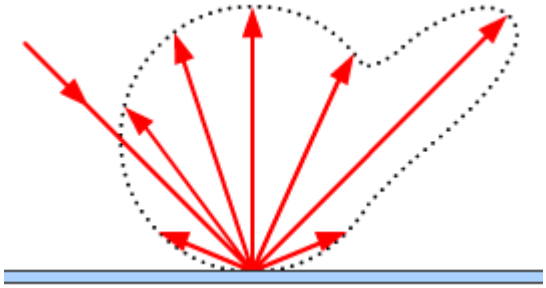


Ilustración 27: Reflexión mixta [WEB05]

Reflectancia espectral

La reflectancia espectral ρ (λ) de una superficie, es la relación entre el flujo radiante incidente y el flujo radiante reflejado en una única longitud de onda en condiciones de geometría fijas (es decir, no cambiantes). Es una magnitud adimensional y se define en porcentajes de 0 a 100% o como factor de 0 a 1. Además tiene en cuenta el flujo radiante, es decir la totalidad de la radiación reflejada por la semiesfera [GUSREF].

La experimentación ha demostrado que el valor de reflectancia espectral no depende de la intensidad o cualidad de la luz incidente, sino que se trata de una propiedad intrínseca de la superficie.

$$\rho(\lambda) = \frac{\Phi_{\lambda\rho}}{\Phi_{\lambda}}$$

Transmitancia

La transmisión o transmitancia τ de la energía radiante a través de un medio es el fenómeno por el cual esa energía lo atraviesa y sale por otro lado. Un cuerpo es opaco si la energía no se transmite o se transmite sólo de una forma muy reducida. Si el cuerpo no es opaco, y por tanto buena parte de la radiación se transmite a través de él [GUSTRA].

Expresada en forma porcentual resulta:

$$\tau(\%) = \frac{\Phi_{\tau}}{\Phi}.100$$

La transmisión o transmitancia puede ser:

- **Transmisión regular o especular**: τ_r cuando sigue las leyes de refracción y no hay difusión.

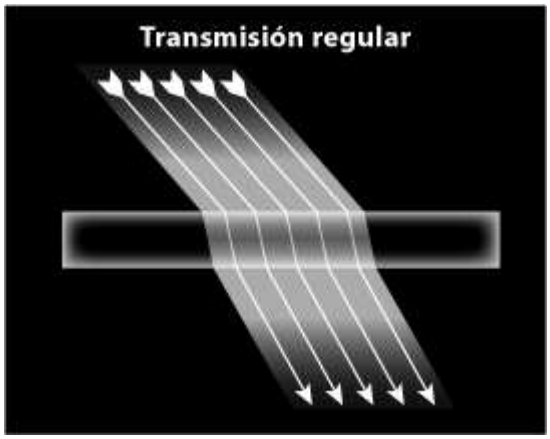


Ilustración 28: Transmisión regular o especular [GUSTRA].

- **Transmisión difusa**: τ_d si la transmisión va acompañada de difusión.

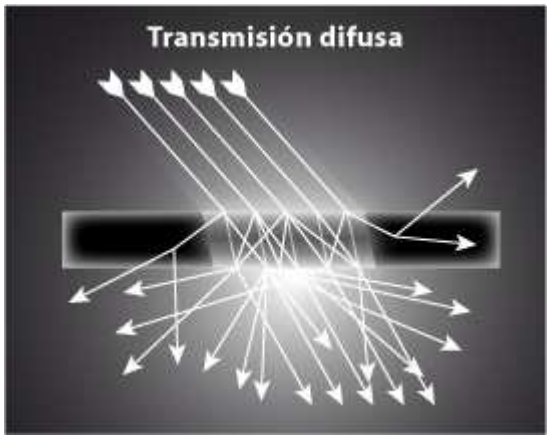


Ilustración 29: Transmisión difusa [GUSTRA].

En cuanto a la transmitancia mixta, tenemos que es una combinación de partes de transmisión regular y difusa donde: $\tau = \tau_r + \tau_d$

Transmitancia espectral

La transmitancia espectral τ (λ) es la relación entre el flujo radiante transmitido y el incidente en una única longitud de onda en condiciones geométricas fijas [GUSTRA].

$$\tau(\lambda) = \frac{\Phi_{\lambda\rho}}{\Phi_{\lambda}}$$

Un cuerpo es transparente si la transmitancia es alta y regular, mientras que es translúcido si la transmitancia es alta y difusa. Para un cuerpo transparente o translúcido, la percepción del color se determina por la selectividad espectral de la absorción:

- Si toda la luz incidente se transmite de forma especular, el cuerpo es transparente e incoloro.

- Si toda la luz incidente se transmite de forma difusa, el cuerpo es translúcido e incoloro.

- Si la luz se absorbe selectivamente y el resto se transmite especularmente, el cuerpo es transparente y coloreado.

- Si la luz se transmite en su mayor parte y el resto se transmite difusamente, el cuerpo es translúcido y coloreado.

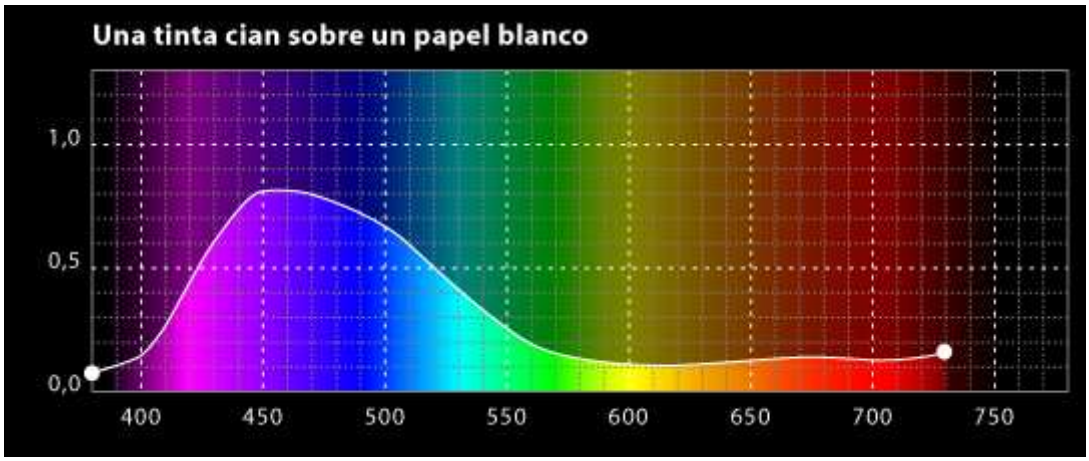


Ilustración 30: Reflectancia espectral de una tinta cyan sobre un papel blanco [GUSTRA].

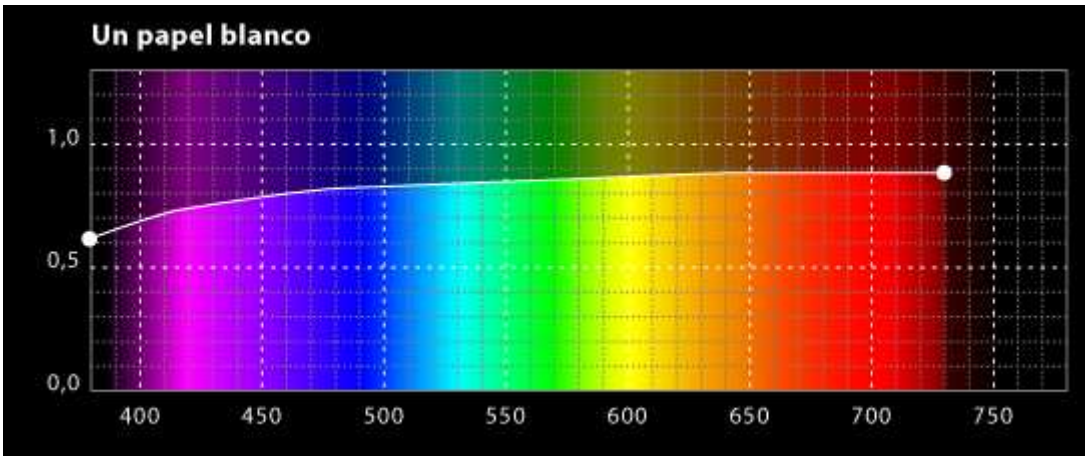


Ilustración 31: Reflectancia espectral de un papel blanco [GUSTRA].

1.7.4 Campo Fisiológico

1.7.4.1 El Ojo

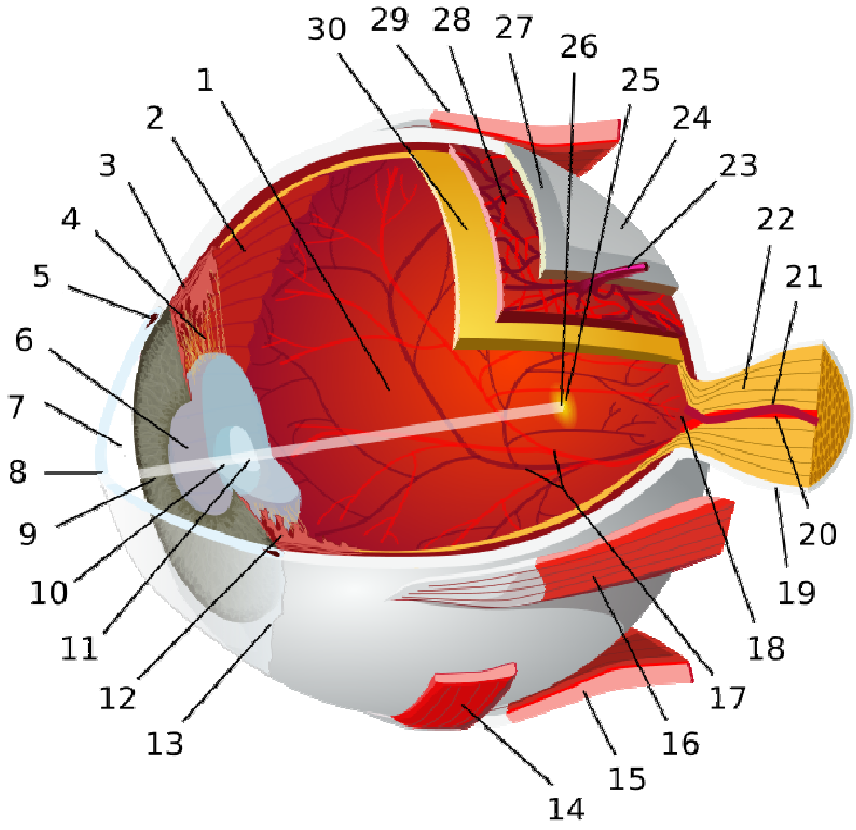


Ilustración 32: Diagrama de un ojo humano [WEB01]

En la Ilustración 32 se aprecian las partes del ojo humano: 1:humor vítreo 2:ora serrata 3:músculo ciliar 4:ligamento suspensorio del lente 5:canal de Schlemm 6:pupila 7:cámara anterior 8:córnea 9:iris 10:cortex del cristalino 11:núcleo del cristalino 12:cuerpo ciliar 13:conjuntiva 14:músculo oblícuo inferior 15:músculo recto inferior 16:músculo recto medial 17:arterias y venas retinianas 18:papila (punto ciego) 19:duramadre 20:arteria central retiniana 21:vena central retiniana 22:nervio óptico 23:vena vorticosa 24:conjuntiva bulbar 25:mácula 26:fóvea 27:esclerótica 28:coroides 29:músculo recto superior 30:retina.
Muchas capacidades del sistema visual pueden ser comprendidas conociendo la organización de la retina. Los dos tipos de fotorreceptores, llamados bastones y conos por su apariencia anatómica, tienen diferentes sensibilidades a la longitud de onda, diferentes sensibilidades absolutas a la luz y poseen diferente distribución en la retina.

Los bastones tienen mayor sensibilidad absoluta a la luz y en consecuencia son los responsables de la visión nocturna. Los conos, menos sensibles a la luz, se clasifican, según su sensibilidad espectral a diferente longitud de onda, en tres tipos diferentes identificados por rojos, verdes y azules, según estén asociados a longitudes de onda largas, medias o cortas. Estos tres tipos de conos son los responsables de la percepción del color [ELIA02].

El sistema visual puede operar sobre un rango de alrededor de 12 unidades logarítmicas, desde una luminancia de 10^{-6} cd/m^2 hasta los 10^{6} cd/m^2. Sin embargo este amplio rango no se cubre simultáneamente, debido a que el sistema visual solo puede cubrir un rango de 2 o 3 unidades logarítmicas de luminancia. Los valores que están por encima de este limitado rango son vistos como deslumbrantes y aquellos valores que estén por debajo quedan oscuros. La Tabla 3 sintetiza las capacidades del sistema visual en cada uno de sus rangos funcionales.

Tabla 3: Rangos funcionales de las capacidades del sistema visual humano [ELIA02].

Nombre	Rango (cd/m^2)	Capacidades	Fotorreceptor activo
Fotópico	> 3	Visión de color. Buena discriminación de detalles.	Conos
Mesópico	> 0.001 y < 3	Visión de color disminuida. Reducida discriminación de detalles. Corrimiento en la sensibilidad espectral.	Conos y bastones
Escotópico	< 0.001	Sin visión de color. Muy pobre discriminación de detalles.	Bastones

El hecho principal que debemos retener en la visión en colores es la Ley de la Trivariancia Visual (Ley de Grassman) que se revisará mas adelante. Con esta ley es posible reproducir cualquier color con la mezcla aditiva e tres colores básicos. Esta es la razón por la cual tres fósforos son suficientes para ver cualquier color en una pantalla cromática de un computador o de un TV [DESC99].

1.7.4.2 Trivariancia Visual

La trivariancia visual es una propiedad fundamental del receptor visual y la podemos ilustrar o demostrar cuando decimos que un color puede ser igualado por la suma de tres colores primarios convenientemente elegidos e independientes entre si. Esta posibilidad de adicionar tres luces necesarias y suficientes para igualar una luz cualquiera es consecuencia de las propiedades fisiológicas del ojo, es decir su trivariancia [DESC99]. Una mezcla de los tres colores primarios rojo, verde, azul (red, green, blue), producen el color blanco y confirman esta trivariancia.

Según la "*Ley de Trivariancia Visual de Grassman*", la trivariancia puede expresarse por medio de:

$$C = \alpha(R) + \beta(G) + \gamma(B)$$

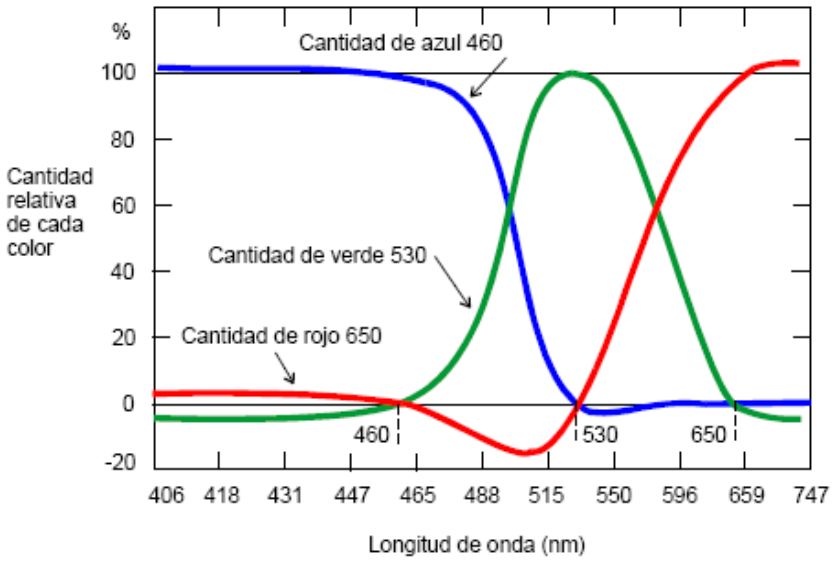


Ilustración 33: Cantidades relativas de tres colores necesarias para igualar cualquier color monocromático.

El trabajo realizado por ELI de Argentina [ELIA02] señala en la Ilustración 33 las cantidades relativas de tres colores (rojo, verde y azul) necesarias para igualar cualquier color monocromático (espectral). Son curvas estandarizadas pues los datos reales varían según el observador y la intensidad de la luz.

1.7.4.2.1 Componentes tricromáticos

Las cantidades de los estímulos de referencia que se deben mezclar para reproducir el color de un estimula dado, se denominan componentes tricromáticos de ese estimulo.

Los cálculos colorimétricos se fundamentan en las Leyes de Grassman [DESC99]:

- Un flujo radiante de una distribución espectral dada $\Phi_e(\lambda)$ [w] que produce una sensación de color [w] C_1 es equivalente (≡) a la suma de los [w] del primario azul A, del primario verde V y del primario rojo R. Los [w] de los primarios son los componentes tricromáticos espectrales de los estímulos de referencia (primarios).

$$c_1 C_1 \equiv a_1 A + v_1 V + r_1 R$$

donde a_1, v_1 y r_1 son componentes tricromáticos de los estímulos de referencia y cuya suma es igual a c_1. Lo esencial de esta ley es que existe una y sólo una combinación de tres primarios que iguala perceptualmente cualquier color real.

- Si se multiplica por un mismo número a la radiación que produce una sensación de color dada, los componentes tricromáticos de la mezcla quedan multiplicados por ese mismo número y la sensación no cambia.

$$n.c_1C_1 \equiv n.a_1A + n.v_1V + n.r_1R$$

- Si se suman dos estímulos de color cualesquiera evaluados por sus componentes tricromáticos, la suma algebraica de ambos es equivalente a la suma algebraica de sus respectivos componentes tricromáticos:

$$c_1C_1 \equiv a_1A + v_1V + r_1R$$

$$c_2C_2 \equiv a_2A + v_2V + r_2R$$

$$\overline{c_1C_1 + c_2C_2 \equiv (a_1 + a_2)A + (v_1 + v_2)V + (r_1 + r_2)R}$$

1.7.4.3 Espacios Colorimétricos C. I. E

1.7.4.3.1 Espacio Colorimétrico R, G, B, 1931

El Espacio Colorimétrico RGB o comúnmente llamado Modelo RGB esta basado directamente en el modelo triestímulos y de síntesis aditiva. Es denominado RGB por (red, green, blue), los colores triestímulos rojo, verde y azul con los cuales se puede reproducir cualquier color mediante adición de sus partes.

El modelo RGB forma un cubo tridimensional como se puede apreciar en la Ilustración 34, de manera que cada eje de este espacio tridimensional representa un color. El rango de cada coordenada o componente cromático RGB suele ser [0, 1], aunque generalmente es el procesamiento de imágenes se utiliza el intervalo [0, 255]. Todas las coordenadas que se extienden en la

línea que parte del punto (0,0,0) al punto (255, 255, 255) corresponden a la escala de grises.

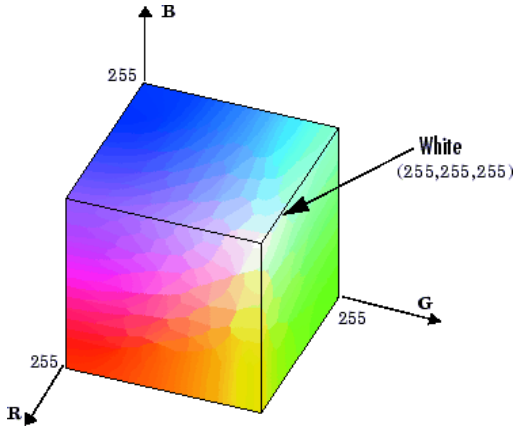


Ilustración 34: El cubo de color RGB [WEB12]

1.7.4.3.2 Espacio Colorimétrico XYZ

Este sistema fue adoptado por la C.I.E. con el fin de eliminar las ordenadas negativas de las curvas colorimétricas (véase Ilustración 35). El nuevo sistema se deriva del RGB aplicando una transformación lineal como se demuestra en [DESC99].

Para la definición del espacio de color XYZ, los coeficientes a utilizar para la transformación lineal fueron elegidos de modo que:

- Los valores triestímulos que conforman la curva colorimétrica no presentan ordenadas negativas en ninguna de las longitudes de onda del espectro.

- Una de esas tres curvas se la hace coincidir con la curva V(λ) de Eficiencia luminosa espectral para el Observador Patrón Fotópico C.I.E.

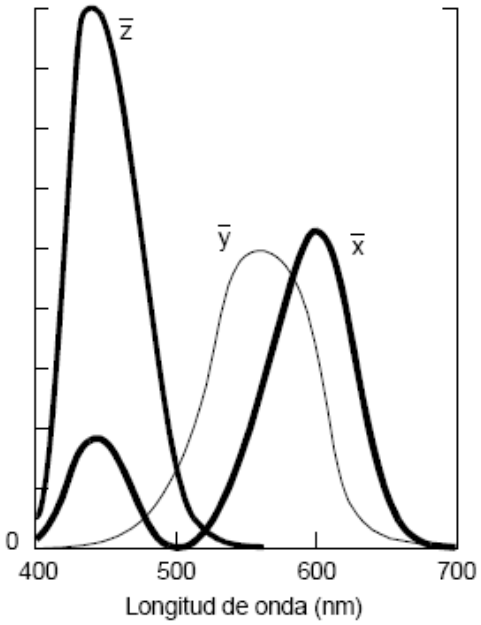


Ilustración 35: Curvas de los valores triestímulos del espectro equienergético del Observador Patrón para Colorimetría C.I.E [DESC99].

Las ordenadas de las tres curvas se las identifica como $\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$ y están tabuladas cada 5 nm en el intervalo entre 380 nm y 700 nm según la C.I.E. La Tabla 4 contiene estos valores.

Tabla 4: Valores triestímulos del espectro equienergético Observador Patrón Normal C.I.E 1931.

λ (nm)	Valores triestímulos Espectrales $\bar{x}(\lambda)$	$\bar{y}(\lambda)$	$\bar{z}(\lambda)$	Coordenadas de Cromaticidad x	y	z
380	0.0014	0.0000	0.0065	0.1741	0.0050	0.8209
385	0.0022	0.0001	0.0105	0.1740	0.0050	0.8210
390	0.0042	0.0001	0.0201	0.1738	0.0049	0.8213
395	0.0076	0.0002	0.0362	0.1736	0.0049	0.8215
400	0.0143	0.0004	0.0679	0.1733	0.0048	0.8219
405	0.0232	0.0006	0.1102	0.1730	0.0048	0.8222
410	0.0435	0.0012	0.2074	0.1726	0.0048	0.8226
415	0.0776	0.0022	0.3713	0.1721	0.0048	0.8231
420	0.1344	0.0040	0.6456	0.1714	0.0051	0.8235
425	0.2148	0.0073	1.0391	0.1703	0.0058	0.8239

430	0.2839	0.0116	1.3856	0.1689	0.0069	0.8242
435	0.3285	0.0168	1.6230	0.1669	0.0086	0.8245
440	0.3483	0.0230	1.7471	0.1644	0.0109	0.8247
445	0.3481	0.0298	1.7826	0.1611	0.0138	0.8251
450	0.3362	0.0380	1.7721	0.1566	0.0177	0.8257
455	0.3187	0.0480	1.7441	0.1510	0.0227	0.8263
460	0.2908	0.0600	1.6692	0.1440	0.0297	0.8263
465	0.2511	0.0739	1.5281	0.1355	0.0399	0.8246
470	0.1954	0.0910	1.2876	0.1241	0.0578	0.8181
475	0.1421	0.1126	1.0419	0.1096	0.0868	0.8036
480	0.0956	0.1390	0.8130	0.0913	0.1327	0.7760
485	0.0580	0.1693	0.6162	0.0687	0.2007	0.7306
490	0.0320	0.2080	0.4652	0.0454	0.2950	0.6596
495	0.0147	0.2586	0.3533	0.0235	0.4127	0.5638
500	0.0049	0.3230	0.2720	0.0082	0.5384	0.4534
505	0.0024	0.4073	0.2123	0.0039	0.6548	0.3413
510	0.0093	0.5030	0.1582	0.0139	0.7502	0.2359
515	0.0291	0.6082	0.1117	0.0389	0.8120	0.1491
520	0.0633	0.7100	0.0782	0.0743	0.8338	0.0919
525	0.1096	0.7932	0.0573	0.1142	0.8262	0.0596
530	0.1655	0.8620	0.0422	0.1547	0.8059	0.0394
535	0.2257	0.9149	0.0298	0.1929	0.7816	0.0255
540	0.2904	0.9540	0.0203	0.2296	0.7543	0.0161
545	0.3597	0.9803	0.0134	0.2658	0.7243	0.0099
550	0.4334	0.9950	0.0087	0.3016	0.6923	0.0061
555	0.5121	1.0000	0.0057	0.3373	0.6589	0.0038
560	0.5945	0.9950	0.0039	0.3731	0.6245	0.0024
565	0.6784	0.9786	0.0027	0.4087	0.5896	0.0017
570	0.7621	0.9520	0.0021	0.4441	0.5547	0.0012
575	0.8425	0.9154	0.0018	0.4788	0.5202	0.0010
580	0.9163	0.8700	0.0017	0.5125	0.4866	0.0009
585	0.9786	0.8163	0.0014	0.5448	0.4544	0.0008
590	1.0263	0.7570	0.0011	0.5752	0.4242	0.0006
595	1.0567	0.6949	0.0010	0.6029	0.3965	0.0006
600	1.0622	0.6310	0.0008	0.6270	0.3725	0.0005
605	1.0456	0.5668	0.0006	0.6482	0.3514	0.0004
610	1.0026	0.5030	0.0003	0.6658	0.3340	0.0002
615	0.9384	0.4412	0.0002	0.6801	0.3197	0.0002
620	0.8544	0.3810	0.0002	0.6915	0.3083	0.0002
625	0.7514	0.3210	0.0001	0.7006	0.2993	0.0001

630 0.6424 0.2650 0.0000 0.7079 0.2920 0.0001
635 0.5419 0.2170 0.0000 0.7140 0.2859 0.0001
640 0.4479 0.1750 0.0000 0.7190 0.2809 0.0001
645 0.3608 0.1382 0.0000 0.7230 0.2770 0.0000
650 0.2835 0.1070 0.0000 0.7260 0.2740 0.0000
655 0.2187 0.0816 0.0000 0.7283 0.2717 0.0000
660 0.1649 0.0610 0.0000 0.7300 0.2700 0.0000
665 0.1212 0.0446 0.0000 0.7311 0.2689 0.0000
670 0.0874 0.0320 0.0000 0.7320 0.2680 0.0000
675 0.0636 0.0232 0.0000 0.7327 0.2673 0.0000
680 0.0468 0.0170 0.0000 0.7334 0.2666 0.0000
685 0.0329 0.0119 0.0000 0.7340 0.2660 0.0000
690 0.0227 0.0082 0.0000 0.7344 0.2656 0.0000
695 0.0158 0.0057 0.0000 0.7346 0.2654 0.0000
700 0.0114 0.0041 0.0000 0.7347 0.2653 0.0000
705 0.0081 0.0029 0.0000 0.7347 0.2653 0.0000
710 0.0058 0.0021 0.0000 0.7347 0.2653 0.0000
715 0.0041 0.0015 0.0000 0.7347 0.2653 0.0000
720 0.0029 0.0010 0.0000 0.7347 0.2653 0.0000
725 0.0020 0.0007 0.0000 0.7347 0.2653 0.0000
730 0.0014 0.0005 0.0000 0.7347 0.2653 0.0000
735 0.0010 0.0004 0.0000 0.7347 0.2653 0.0000
740 0.0007 0.0002 0.0000 0.7347 0.2653 0.0000
745 0.0005 0.0002 0.0000 0.7347 0.2653 0.0000
750 0.0003 0.0001 0.0000 0.7347 0.2653 0.0000
755 0.0002 0.0001 0.0000 0.7347 0.2653 0.0000
760 0.0002 0.0001 0.0000 0.7347 0.2653 0.0000
765 0.0001 0.0000 0.0000 0.7347 0.2653 0.0000
770 0.0001 0.0000 0.0000 0.7347 0.2653 0.0000
775 0.0001 0.0000 0.0000 0.7347 0.2653 0.0000
780 0.0000 0.0000 0.0000 0.7347 0.2653 0.0000

1.7.4.3.2.1 Cálculo de los Valores Triestímulos XYZ

Una vez obtenidas las curvas de los valores triestímulos del espectro equienergético del Observador Patrón para Calorimetría C.I.E., el calculo de un estimulo de color es bastante simple. Aplicando el principio de aditividad formulado por Grassman [DESC99] se obtiene:

$$X = k \int_{360}^{830} \Phi_{\lambda}(\lambda).\bar{x}(\lambda).d\lambda$$

$$Y = k \int_{360}^{830} \Phi_{\lambda}(\lambda).\bar{y}(\lambda).d\lambda$$

$$Z = k \int_{360}^{830} \Phi_{\lambda}(\lambda).\bar{z}(\lambda).d\lambda$$

donde: $\Phi_{\lambda}(\lambda)$ es la función espectral que representa el estimulo de color que se desea evaluar.

$\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$ son los valores triestímulos de color C de radiación $\Phi_{\lambda}(\lambda)$.

k es una constante de normalización de valor arbitrario.

Para fines prácticos, la integración se puede realizar mediante sumas numéricas en intervalos de longitud de onda $\Delta\lambda = 5$ nm, de manera que las expresiones anteriores quedan expresadas como:

$$X = k\sum_{360}^{830} \Phi_{\lambda}(\lambda).\bar{x}(\lambda).\Delta\lambda \qquad Y = k\sum_{360}^{830} \Phi_{\lambda}(\lambda).\bar{y}(\lambda).\Delta\lambda \qquad Z = k\sum_{360}^{830} \Phi_{\lambda}(\lambda).\bar{y}(\lambda).\Delta\lambda$$

1.7.4.3.2.2Cálculo de las Coordenadas de Cromaticidad

Según [DESC99], las coordenadas de cromaticidad se obtienen de los valores triestímulos XYZ:

$$x = \frac{X}{X+Y+Z} \qquad y = \frac{Y}{X+Y+Z} \qquad z = \frac{Z}{X+Y+Z}$$

De manera que x + y + z = 1, entonces, z = 1- x –y, por lo tanto se pueden expresar en un plano de dos dimensiones x y. Además se puede notar que los valores de las coordenadas de cromaticidad varían entre 1 y 0.

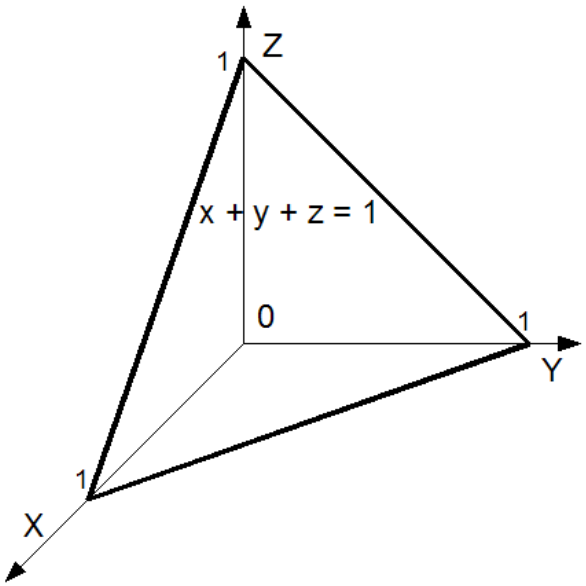


Ilustración 36: Representación grafica del plano x + y + z = 1

1.7.4.3.2.3Diagrama de Cromaticidad

El diagrama de cromaticidad es un buen método de especificación de colores. Todos los colores espectrales saturados al 100% se sitúan al borde del diagrama. Las posiciones de los diferentes colores espectrales (desde el violeta de 380 nm al rojo 780 nm) se indican alrededor de la linea curva. La coordenada E representa al blanco equienergético o espectro equienergético [DESC99], el cual es el punto de igual energía para los tres colores primarios y su saturación es 0.

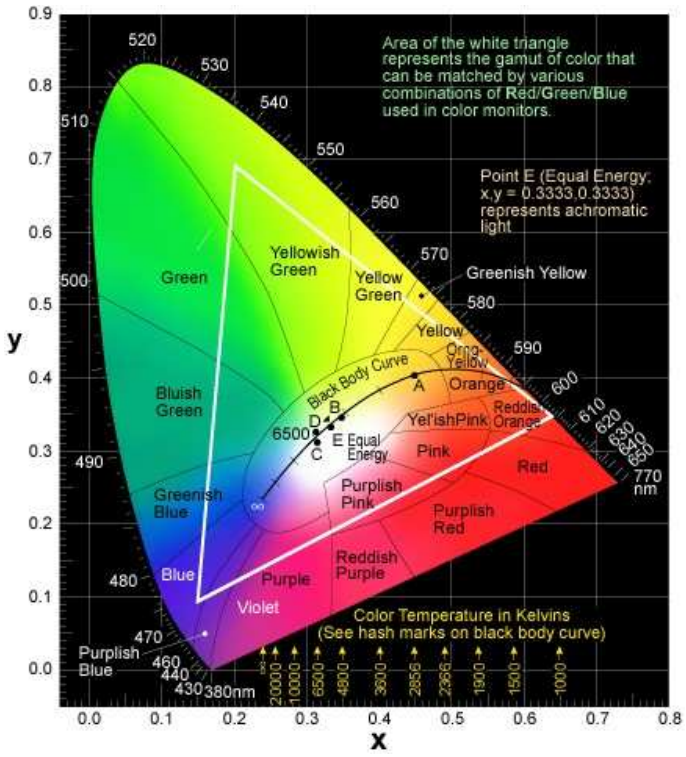


Ilustración 37: Diagrama de cromaticidad C.I.E 1931 [WEB07]

1.7.4.3.2.4Metamerismo y Constancia de Color

Constancia de Color

La constancia del color es un fenómeno de la percepción del color por el que la mayoría de las superficies de color parecen mantener la apariencia cromática que tendrían bajo lo que sería la luz del día, incluso bajo condiciones luminosas muy diferentes a dicho tipo de iluminación. La constancia del color es un poco sorprendente, ya que la distribución espectral de la luz que llega al ojo desde una superficie puede variar extremadamente según cuál sea la fuente de luz.

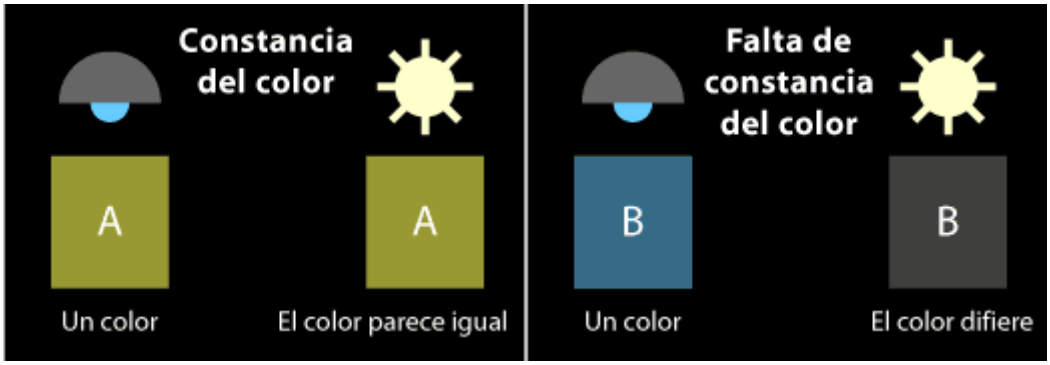


Ilustración 38: La constancia de color y la falta de constancia de color [WEB06].

Sin embargo, el fenómeno de la constancia del color no se da en todos los casos, ya que las superficies no conservan su apariencia de estar bajo una iluminación diurna si se hallan bajo algunos tipos de luces fluorescentes o bajo radicaciones monocromáticas. De hecho, algunas superficies parecen cambiar claramente de aspecto según la fuente de luz bajo la que se hallen. De ese tipo de objetos, se dice que carecen de constancia del color [WEB06].

Metamerismo

El término metamerismo se refiere a la situación en la que dos muestra de color parecen ser iguales en una situación dada y diferentes en otras. En esos casos se dice que hay una correspondencia cromática condicional.

Según [DESC99], "*el ojo no puede decirnos nada respecto a la composición espectral de la luz que le produce la sensación de color*". A cada composición espectral le corresponde un color, pero un color puede ser obtenido por infinitas composiciones espectrales. La medición tendrá que tener en cuenta esta relación unívoca que denominamos "metamerismo".

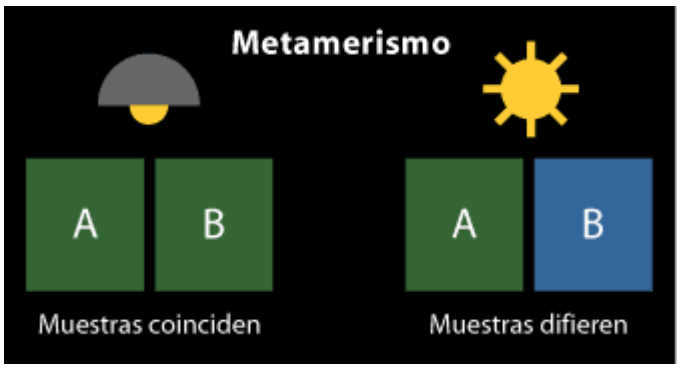


Ilustración 39: El metamerismo [WEB06].

El metamerismo se suele tratar en términos de dos iluminantes (metamerismo del iluminante), donde dos muestras de color parecen ser iguales bajo un iluminante pero no bajo otro. Además, hay otros tipos de metamerismo, como el metamerismo geométrico o el metamerismo del observador. De dos muestras de color que son iguales sólo en ciertas circunstancias se dice que forman un par metamérico [WEB06].

1.7.4.3.2.5Temperatura de Color

De acuerdo a la temperatura T (ºK), la curva de distribución espectral de la exitancia radiante será como la indicada en la Ilustración 40.

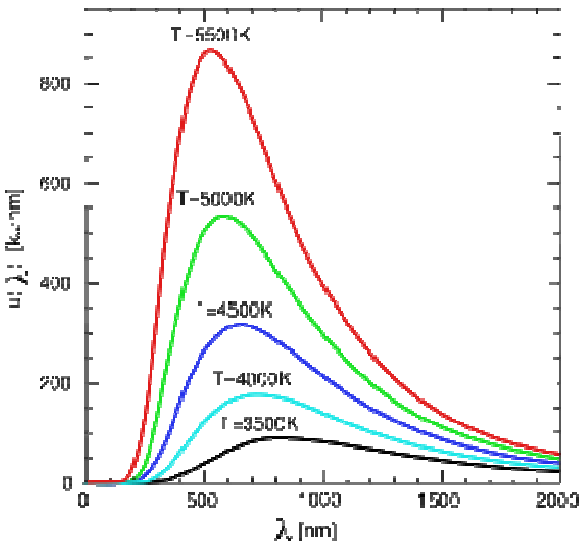


Ilustración 40: Espectro de radiación de un cuerpo negro [WEB01].

Es posible hallar las coordenadas de cromaticidad de las distribuciones espectrales del cuerpo negro en una amplia gama de temperaturas y luego trasladarlas al diagrama de cromaticidad como se explica en [DESC99] y como se aprecia en la Ilustración 37 que muestra algunas temperaturas de color.

En resumen, la temperatura de color describe la apariencia de color de las fuentes luminosas. En la Ilustración 41 se muestra la ubicación de los colores de fuentes incandescentes para distintas temperaturas y se muestran las ubicaciones de tres fuentes estándar blancas: A: incandescente de tungsteno (2854 ºK), B: luz solar al mediodía (4870 ºK) y C: filamento de tungsteno filtrado a “luz dia” (6770 ºK). Según la E.L.I. Argentina [ELIA02], la temperatura de color es una manera conveniente de estandarizar las fuentes de luz, por ejemplo, las fuentes con temperaturas de color altas, mayores de 5000 ºK se consideran de bajo rendimiento de color y se consideran como fuentes “frías”, como las lámparas fluorescentes luz día. En contraste, aquellas con temperaturas menores a 3000 ºK son fuentes “cálidas” como las lámparas incandescentes.

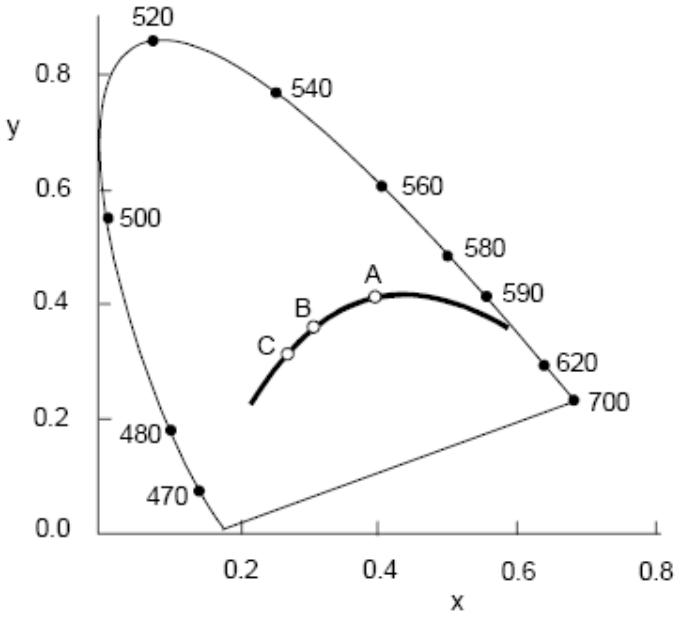


Ilustración 41: Ubicación de distintas fuentes luminosas en el espacio de color [ELIA02].

1.7.4.3.3 Sistema Suplementario Colorimétrico C.I.E 1964

El Sistema de Observador Patrón para Colorimetría que hemos utilizado y que rige desde 1931 fue objetado cuando hubo la necesidad de predecir para un campo mayor que el foveal. Esto corresponde a un campo visual de 2º. En 1964 la C.I.E. estableció un Observador Patrón Suplementario para un campo visual de 10º [DESC99].

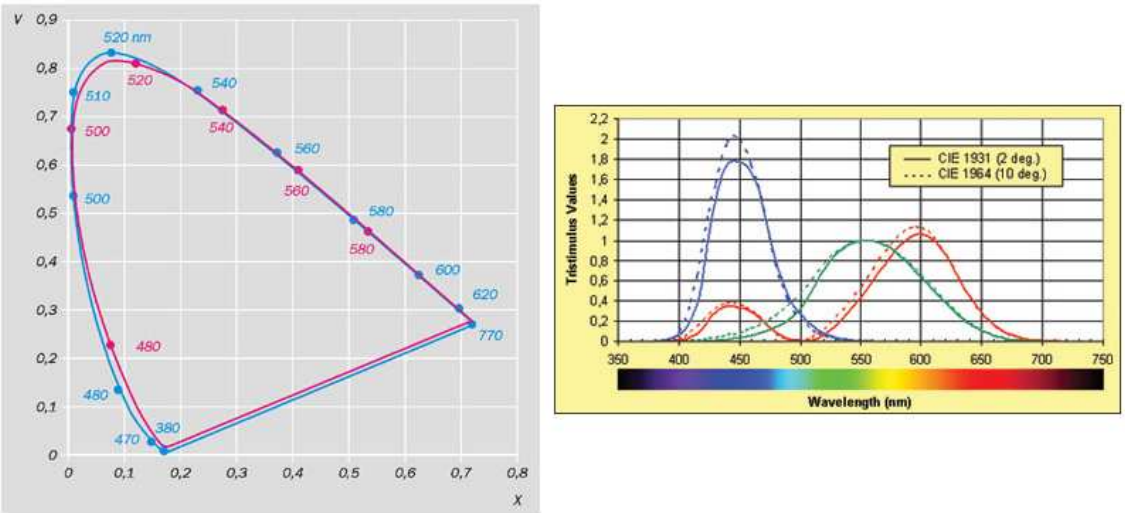


Ilustración 42: Curvas de valores triestímulos y diagramas de cromaticidad para los observadores C.I.E 1931 y 1964.

1.7.4.4 Los Espacios de Color Uniforme de la C.I.E.

Cuando hablamos de diferencia de color, es necesario precisar un poco más las cosas. Si tenemos que comparar dos objetos coloreados diferentes, sus diferencias pueden estar ligadas a su naturaleza y a su textura. Por ejemplo al comparar una tela con una cerámica del mismo color.

De acuerdo a [DESC99], si solo nos limitamos a la comparación entre dos objetos de la misma naturaleza, ocurre muy a menudo que en ángulo desde el cual son vistos afecta la percepción del color. Este comportamiento esta generalmente relacionado a la textura superficial y a las condiciones de iluminación.

Dentro de la discriminación de colores, tenemos el umbral de detección y las tolerancias, parámetros que motivaron a los investigadores del espacio de color de la C.I.E a ingeniarse medidas de diferenciación de color. Es asi que David Mac Adam y Brown encontraron la *mínima diferencia perceptible* de cada color en el diagrama de cromaticidad representadas como elipses. Estas elipses indican cuanto es necesario apartarse de un punto de color para tener la misma diferencia perceptible [DESC99].

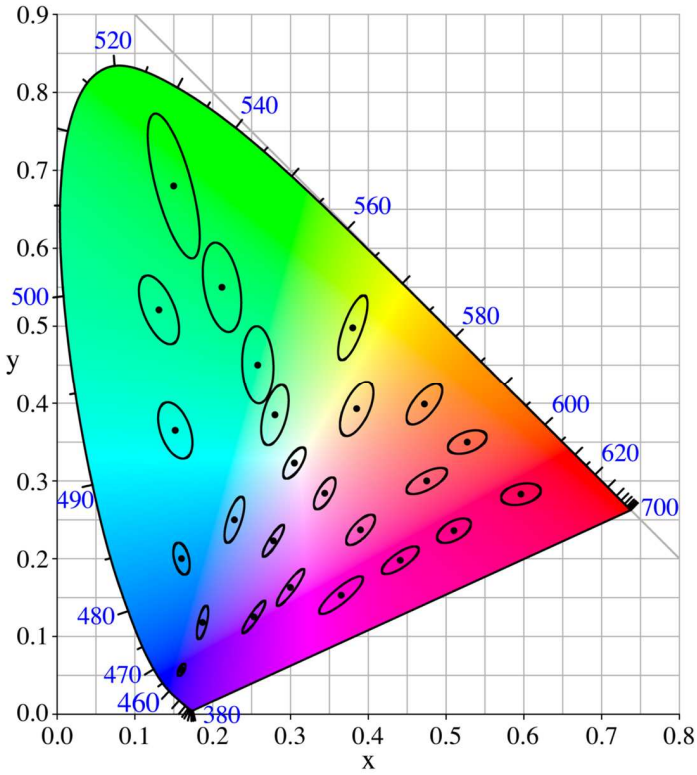


Ilustración 43: Elipses de Mac Adam y Brown (1949) reproducidas sobre un diagrama de cromaticidad C.I.E. 1931. Ampliadas 10 veces según G. Wyszecki y W. S. Stiles [WEB01].

1.7.4.4.1 Diagrama UCS C.I.E 1960 1976

Mac Adam (1937) propuso una transformación lineal del espacio de color la cual fue adoptada por la C.I.E en 1960 y se conoce como Diagrama U.C.S C.I.E (Uniform Chromaticy Scale). La Ilustración 44 muestra como quedó transformado el Diagrama C.I.E. 1931 luego de aplicar las siguientes ecuaciones en función de las coordenadas de cromaticidad (x; y) o los valores triestímulos XYZ C.I.E 1931 [DESC99]:

$$u = \frac{4x}{-2x+12y+3} = \frac{4X}{X+15Y+3Z}$$

$$v = \frac{6y}{-2x+12y+3} = \frac{6Y}{X+15Y+3Z}$$

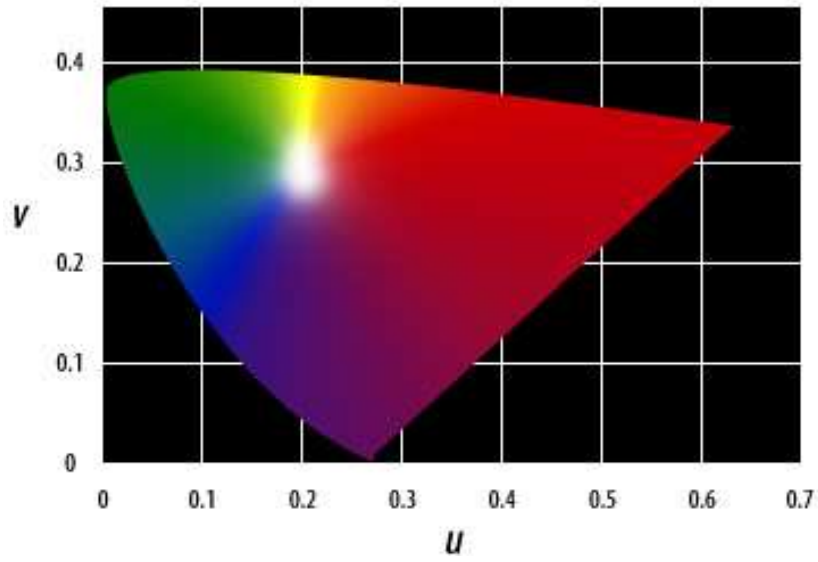


Ilustración 44: Diagrama de Cromaticidad Uniforme UCS - CIE 1960, según Wyszecki y Stiles.

En este nuevo diagrama, la diferencia entre dos cromaticidades vecinas (u_1; v_1) y (u_2; v_2) resulta de:

$$\Delta E = [(u_2 - u_1)^2 + (v_2 - v_1)^2]^{1/2}$$

Para hacer el espacio cromático mas uniforme y posibilitar el cálculo del apartamiento de colores, la C.I.E. recomendó provisoriamente un sistema U*, V*, W* 1964 en el cual segmentos iguales representan sensiblemente diferencias de color juzgadas para condiciones de observación definidas (propuesto por Wyszecki). Finalmente en 1975 el Comité de Colorimetría de la C.I.E. aprobó la adopción de dos nuevos espacios de color y sus respectivas formulaciones para el cálculo de la diferencia de color [DESC99]:

- Espacio de color L*u*v* C.I.E. 1976 (Espacio CIELUV)
- Espacio de color L*a*b* C.I.E. 1976 (Espacio CIELAB)

Conjuntamente con estos espacios se definieron nuevas coordenadas de cromaticidad UCS – CIE 1976 declarando obsoletas las del UCS 1960:

$$u' = u \qquad\qquad v' = \frac{3}{2}v$$

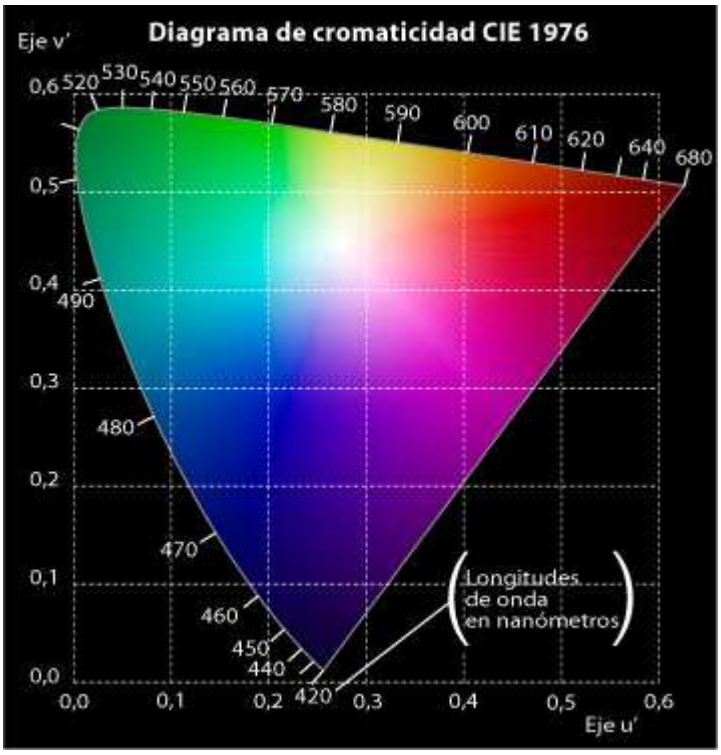


Ilustración 45: Diagrama de Cromaticidad Uniforme UCS- CIE 1976 [WEB06].

El espacio de color CIELUV es una extensión a tres dimensiones de los resultados de Mac Adam basado en una evaluación estadística de las diferencias perceptibles de igualación visual de luces coloreadas [DESC99]. Como mejora del U*V*W* y como toda transformación proyectiva es equivalente a una nueva elección de primarios y las formulas de transformación tienen una forma homográfica bien conocida dada a continuación:

$$U^* = \frac{4X}{9} \qquad V^* = Y \qquad W^* = \frac{-X + 2Y + Z}{3}$$

$$u' = u \qquad v' = \frac{3}{2v}$$

$$L^* = 116(Y/Y_n)^{1/3} - 16 \qquad Y/Y_n > 0.008856$$

$$u^* = 13L^*(u' - u'_n)$$

$$v^* = 13L^*(v' - v'_n)$$

La diferencia entre dos estímulos de color se calcula con la distancia euclidiana entre los puntos representados en el espacio:

$$\Delta E^*_{uv} = [(\Delta L^*)^2 + (\Delta u^*)^2 + (\Delta v^*)^2]^{1/2}$$

Generalmente se utiliza el espacio de Color CIELUV para el estudio de luces, análisis de imágenes y evaluaciones en la diferencia de color. Por ejemplo en un estudio de televisión, donde se puede aprovechar las propiedades aditivas del sistema, mientras que se reserva el CIELAB para el estudio de los colores de las superficies.

1.7.4.5 Reproducción del Color

La visión de los colores debe tener en cuenta condiciones mas reales que las definiciones de un observador patrón fotométrico y colorimétrico. Entre estos mecanismos de la visión, juega un papel muy importante la adaptación cromática del observador. Por ejemplo, después de unos minutos de adaptación, una tela o papel, podrían parecer blancos aunque estén iluminados por una vela, una lámpara incandescente o una fluorescente [DESC99].

Sin embargo a pesar de esta propiedad de adaptación donde vemos una cierta constancia en el color percibido del objeto bajo fuentes diferentes, se presentan también casos donde a pesar de la adaptación, el cambio de fuente genera distorsiones importantes en el color percibido del mismo objeto. Se dice que estos objetos son inestables.

Según [DESC99], la distorsión del color ocurre cuando un objeto es observado bajo diferentes fuentes de luz y se puede expresar por tres caminos:

- **Distorsión colorimétrica:** es la diferencia entre el color (luminancia y cromaticidad, es decir Y xy del Espacio de Color Yxy para Fuentes Secundarias [DESC99]) de un objeto iluminado por una fuente que deseamos ensayar y el color del mismo objeto iluminado por un iluminante normalizado CIE.

- **Distorsión por adaptación cromática:** es la diferencia percibida en el color del objeto debida solamente a la adaptación cromática.

- **Distorsión total del color:** es la diferencia percibida entre el color de un objeto iluminado por una fuente que deseamos ensayar, y la del mismo objeto iluminado por un iluminante normalizado (iluminante CIE) bajo condiciones visuales especificas. Estas condiciones son usualmente que el observador tenga una visión normal de los colores y que esté adaptado al medio circundante iluminado por cada fuente a su respectivo turno.

1.7.4.5.1 IRC Índice de Reproducción de Color C.I.E. 1963

La C.I.E. recomienda un método de ensayo para medir y especificar las propiedades de reproducción de color de una fuente primaria sobre un objeto basado en la distorsión del color percibido por un observador. Clasifica de este modo a las lámparas bajo un número denominado IRC (índice de reproducción de color) que se limita al grado de distorsión colorimétrica [DESC99].

La E.L.I. Argentina [ELIA02], afirma que las distintas fuentes de luz emiten con composiciones espectrales diferentes, por lo tanto, tienen un rendimiento de color diferente (IRC). Para asegurar una buena discriminación de color es necesario usar una fuente de luz que tenga, no solamente un IRC alto, sino además que produzca luz suficiente para asegurar que el sistema visual opere en la región fotópica. Sin embargo, es importante notar que dos fuentes de luz pueden tener el mismo IRC y no reproducir los colores de la misma manera. Por ejemplo, una lámpara incandescente y una fluorescente, ambas con el mismo índice (por ejemplo 90), hacen que los colores azul y verde parezcan diferentes. Por lo tanto, para asegurar una buena apariencia de color tanto como buena discriminación de color se necesita no solamente un IRC alto sino también una fuente de luz intensa.

1.7.4.6 Espacio de Color L*a*b* C.I.E. 1976 CIELAB

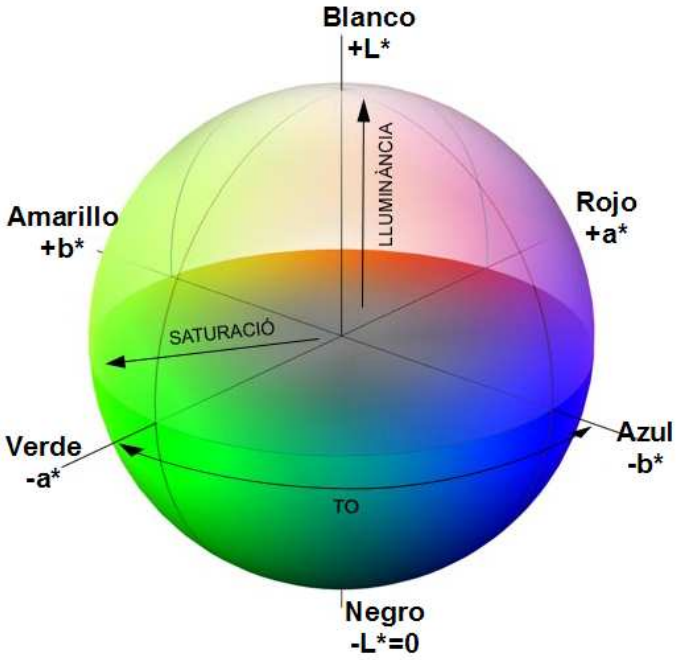


Ilustración 46: Sólido de Color para el Espacio L*a*b* C.I.E. [WEB15]

Este sistema tiene un origen muy diferente al sistema CIELUV. Es consecuencia de los trabajos de E.Q. Adams y D. Nickerson entre 1936 y 1942 y se aplica fundamentalmente a los colores de superficie. Inicialmente, tres valores fueron determinados con la ayuda de la función de Valor de Munsell [DESC99]. Luego estos valores se multiplicaron por el coeficiente 40 y la formula de la diferencia de color fue denominada ANLAB 40, donde la "A" es de Adams, la "N" de Dorothy Nickerson, y "LAB" por los tres ejes [WEB01]. Escribiendo esta formula bajo una forma más cómoda, por sustitución de raíces cúbicas a la función de Valor de Munsell, se obtienen las relaciones del sistema CIELAB. Las formulas de CIELAB son:

Luminosidad variable L*:

$$L^* = 116(Y/Y_n)^{1/3} - 16 \qquad Y/Y_n \geq 0.008856$$

Coordenadas de cromaticidad a* y b*:

$$a^* = 500[(X/X_n)^{1/3} - (Y/Y_n)^{1/3}] \qquad X/X_n \text{ y } Y/Y_n \geq 0.008856$$

$$b^* = 200[(Y/Y_n)^{1/3} - (Z/Z_n)^{1/3}] \qquad Y/Y_n \text{ y } Z/Z_n \geq 0.008856$$

Donde X, Y, Z son los valores triestímulos del estimulo de color considerado. X_n, Y_n, Z_n son los valores triestímulos de un iluminante CIE o el estimulo de color de un objeto blanco especificado. En el caso de que X/X_n, Y/Y_n, Z/Z_n sean menores que 0.008856 se cambian las relaciones de las ecuaciones como se muestra en [DESC99]:

$(X / X_n)^{1/3}$ se reemplaza por $7.787(X / X_n) + 16/116$

$(Y / Y_n)^{1/3}$ se reemplaza por $7.787(Y / Y_n) + 16/116$

$(Z / Z_n)^{1/3}$ se reemplaza por $7.787(Z / Z_n) + 16/116$

Suponiendo dos estímulos de color (véase Ilustración 47): S_1 en un plano de luminosidad constante L* = 85 y S_2 en el plano de luminosidad constante L* = 50. Cada estimulo tiene sus respectivas coordenadas de cromaticidad (a^*_1; b^*_1) y (a^*_2; b^*_2). La diferencia de color queda expresada por la distancia geométrica euclidiana que separa ambos puntos aplicando el teorema de Pitágoras [DESC99]:

$$\Delta E^*_{ab} = [(\Delta L^*)^2 + (\Delta a^*)^2 + (\Delta b^*)^2]^{1/2}$$

$$\Delta E^*_{ab} = [(L^*_2 - L^*_1)^2 + (a^*_2 - a^*_1)^2 + (b^*_2 - b^*_1)^2]^{1/2}$$

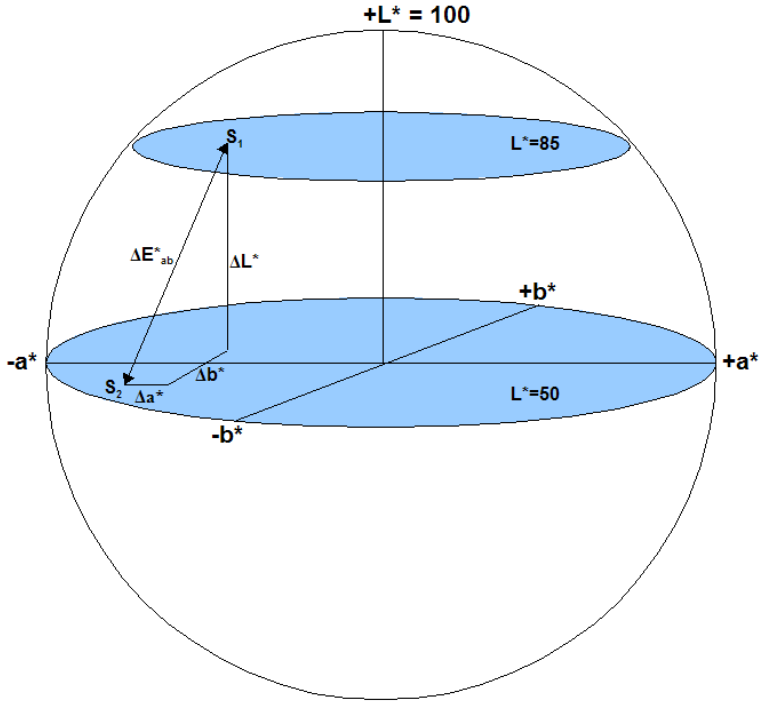


Ilustración 47: Dos estímulos de color S_1 y S_2 representados en el Espacio CIELAB.

1.8 Enfermedades Visuales

1.8.1 Introduccion

La falta de confort visual puede dar origen a una gran variedad de síntomas: enrojecimiento, inflamación, picazón, hormigueo y lagrimeo de los ojos; dolores de cabeza y migrañas; problemas gastrointestinales; dolores y molestias asociados con una mala postura, etc. [ELIA02]. Estos síntomas pueden indicar que el trabajador padece una de varias enfermedades laborales relacionadas con un ambiente mal iluminado.

El trabajo diario con pantallas de visualización de datos (PVD) o monitores de computador puede agravar las anomalías ya existentes en la vista y aumentar la tensión del ojo cuando no funciona correctamente, ocasionando fatiga visual u otros problemas físicos y psicológicos. Por esta razón los operadores de PVD deben acudir regularmente a exámenes oftalmológicos [BERT00].

A continuación se describen las enfermedades y defectos visuales más comunes:

1.8.2 Asthenopia

Es el término médico que designa la fatiga visual. Abarca todos los síntomas asociados con el esfuerzo muscular excesivo efectuado por los ojos durante un período importante, puede resultar difícil de distinguir de los síntomas producidos por el cansancio físico y mental. Los tres síntomas asociados a la asthenopia son:

- **Síntomas oculares** (por ejemplo, sensación de quemaduras, escozor de los globos oculares, mayor sensibilidad, enrojecimiento de los ojos).

- **Síntomas visuales** (por ejemplo, dificultad para enfocar, visión borrosa, manchas delante de los ojos, sensibilidad a la luz, doble visión).

- **Síntomas generales** (por ejemplo, dolores de cabeza, vértigos, náuseas, dolores cervicales, dorsales).

Estos síntomas se manifiestan generalmente por la noche. Las causas más comunes de la fatiga ocular son:

- La obligación de concentrarse por un largo periodo en un objeto fijo sin relajar él mecanismo de acomodación o hacer un numero cada vez mayor de movimientos de acomodación en un tiempo determinado

- Los pasos de la luz natural a la artificial o viceversa, poniendo en evidencia los defectos oculares existentes.

- El pasar la lectura de una imagen normal y uniforme a una imagen que contenga centelleo, oscilaciones y movimiento incontrolado de la imagen de la pantalla.

1.8.3 Hipermetropía

Es la dificultad para ver objetos que están cerca y es el resultado de la imagen visual que se enfoca por detrás de la retina, en lugar de ser directamente sobre ésta, y puede ser causada por el hecho de que el globo ocular es demasiado pequeño o que el poder de enfoque es demasiado débil.

La hipermetropía con frecuencia está presente desde el nacimiento, pero los niños tienen un cristalino muy flexible, que los ayuda a compensar el problema. La mayoría de los niños superan esta afección con el tiempo. A medida que se presenta el envejecimiento, es posible que se requiera el uso de gafas o lentes de contacto para corregir la visión. En caso de tener familiares con hipermetropía, la persona tiene mayor probabilidad de padecer este problema.

Los principales síntomas de la hipermetropía son:

- Visión borrosa de objetos cercanos
- Fatiga ocular
- Dolor ocular
- Dolor de cabeza al leer

1.8.4 Miopía

En la miopía los ojos no enfocan correctamente, haciendo que los objetos distantes se vean borrosos. Con este trastorno, los objetos que están cerca pueden verse claramente, mientras que los que se encuentran a distancia se ven borrosos. Como resultado, la persona con miopía tiende a torcer la vista cuando observa objetos distantes y es la característica base de la palabra *miopía* que proviene de dos palabras griegas: *myein* que significa cerrar y *ops* que significa ojo.

Esta condición es el resultado de centrar la imagen visual delante de la retina en vez de hacerlo directamente en ella. Se presenta cuando la longitud física del ojo es mayor que la longitud óptica. La miopía afecta a hombres y mujeres por igual y los antecedentes familiares de visión corta constituyen un factor de riesgo para desarrollarla. La mayoría de los ojos con miopía son completamente sanos, pero algunas personas desarrollan una forma de degeneración retinal.

Los principales síntomas de la miopía son:

- Visión borrosa o torcer los ojos al tratar de mirar objetos distantes.
- Tensión ocular
- Dolores de cabeza

1.8.5 Presbicia

Es una afección en la cual el cristalino del ojo pierde su capacidad para enfocar, lo que dificulta el hecho de ver objetos cercanos. La afección está asociada con el envejecimiento y empeora con el paso del tiempo. El poder de enfoque del ojo depende de la elasticidad del cristalino, la cual gradualmente se pierde a medida que las personas envejecen. El resultado es una disminución lenta de la capacidad del ojo para enfocar los objetos cercanos.

Es un problema común entre los operadores de PVD a partir de los 30 años y que se acrecienta a partir de las personas mayores de 40 años. La presbicia

también es una parte natural del proceso de envejecimiento y afecta a todas las personas.

Los principales síntomas de la presbicia son:

- Disminución en la capacidad para enfocar objetos cercanos
- Fatiga ocular
- Dolor de cabeza

1.8.6 Glaucoma

Esta es una enfermedad que se produce como consecuencia de varios factores, afectando generalmente a adultos con más de 30 años de edad, es de carácter hereditario pero también se produce como causa de un golpe en los ojos o de tensiones emocionales. En el caso que existan antecedentes de enfermedad en la familia del operador, este debe consultar de modo preventivo a un facultativo especializado.

En el área de trabajo con PVD, se da como consecuencia de la presión del trabajo (tensión emocional) a veces combinada con factores hereditarios. El glaucoma es una enfermedad progresiva que generalmente en sus comienzos no presenta síntomas visibles y cuando estos aparecen la visión ya ha sido afectada.

La enfermedad se produce dentro del ojo; entre el cristalino y la córnea, hay un liquido transparente que se produce y elimina continuamente; si los canales de salida de este líquido se cierran, se dificulta o bloquea la eliminación del mismo, produciéndose su acumulación, trayendo el aumento de presión dentro del ojo, que hace disminuir el flujo de sangre al nervio óptico, ocasionando daño.

Síntomas del glaucoma de ángulo abierto:

- Pérdida gradual de la visión periférica (lateral)
- La mayoría de las personas son asintomáticas hasta que pierden la visión

Síntomas del glaucoma de ángulo cerrado:

- Visión borrosa o disminuida
- Náuseas y vómitos
- Pupila no reactiva a la luz
- Enrojecimiento de los ojos
- Dolor ocular intenso, dolor facial
- Inflamación del ojo

Síntomas del glaucoma congénito:

- Opacidad en la parte frontal del ojo
- Agrandamiento de uno o ambos ojos
- Enrojecimiento de los ojos
- Sensibilidad a la luz
- Lagrimeo

1.9 Fuentes Luminosas

1.9.1 Introduccion

Una de las primeras formas de iluminación artificial utilizadas por el hombre fueron aquellas basadas en algún tipo de combustión: el fuego, las antorchas, las velas, etc. El fuego le permitió al hombre primitivo no solo una fuente de luz, sino también una fuente de calor, seguridad y una manera de preparar mejor sus alimentos al cocinarlos. Más tarde se empelaron lámparas de combustión de aceites y ceras, las cuales eran portables y posteriormente servían para iluminar espacios más grandes.

Alrededor del siglo XVIII se iluminaban las calles con lámparas de gas que se caracterizaban por su peculiar parpadeo. Posteriormente, ya hacia el final del siglo XIV, la revolución industrial y el descubrimiento de la electricidad incentivó la carrera por encontrar una manera segura de iluminación eléctrica. Fue entonces que el inventor estadounidense Thomas Alva Edison[14] creó en 1879 la bombilla incandescente, un éxito comercial que revolucionó la manera de iluminación de casas, oficinas y posteriormente toda América. La patente de Edison sigue siendo el modelo base desde el cual nuevas y mejores fuentes luminosas alumbran nuestros hogares y lugares de trabajo.

1.9.2 Gráficos y Diagramas

1.9.2.1 Diagrama polar o curvas de distribución luminosa

Toda fuente de luz tendrá su indicatrix[15] de intensidad luminosa. En la práctica de la luminotecnia se ha adoptado la presentación de esta información por medio de semiplanos que rotan alrededor de un eje que pasa por el centro óptico de la fuente [DESC99]. Se establecen de este modo sistemas de coordenadas espaciales como los denominados:
(A; α), (B; β) y (C; γ).

[14] Thomas Alva Edison (1847 – 1931 EEUU) Considerado como una de las mentes inventoras más importantes del siglo XX con más de 1000 patentes a su nombre.
[15] Indicatrix es la superficie determinada por la extremidad del vector de intensidad en todo el espacio.

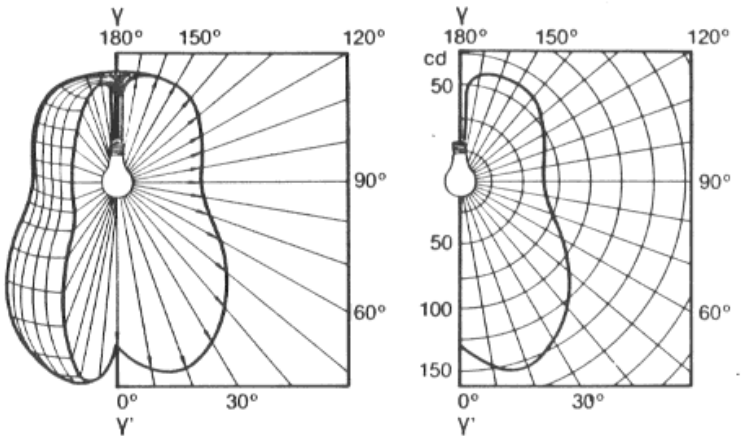


Ilustración 48: Indicatrix de intensidad luminosa lámpara incandescente (izq.). Curva de distribución luminosa (der.) [DESC99].

La Ilustración 48 (izq.) representa la indicatrix de una lámpara incandescente de bulbo claro y uno de los sistemas de coordenadas (C; γ) con la vista de la posición de un semiplano C. La Ilustración 48 (der.) muestra dicho semiplano con la posición de las intensidades luminosas en función de los ángulos γ desde 0º hasta 180º. La intersección de un semiplano C con la indicatrix de intensidades determina la curva de distribución luminosa para esa posición del semiplano que puede rotar de 0º a 360º. Fijando un origen para esta rotación, la intensidad luminosa se expresará como I_v (C; γ).

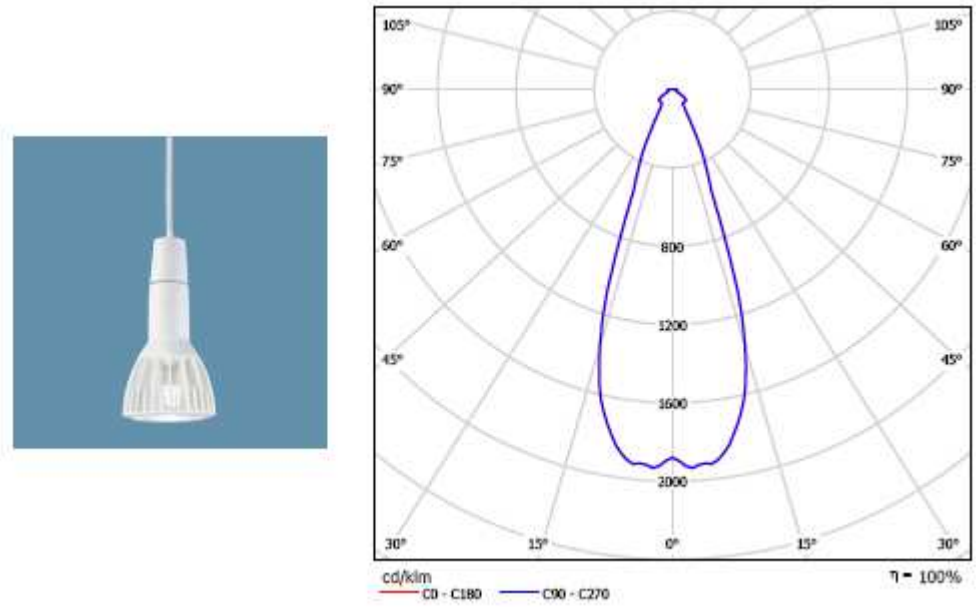


Ilustración 49: Curva de distribución luminosa (OSRAM GmbH 41473 DECOSTAR® STUDIO IRC-P 35W)

Si la indicatrix corresponde a una fuente simétrica como la del ejemplo, bastará conocer las intensidades en uno de los semiplanos C. Si la indicatrix es asimétrica, se elegirán tantos semiplanos C como sean necesarios. En la practica se consideran como suficientes hasta tres semiplanos: C = 0º, C = 45º y C = 90º.

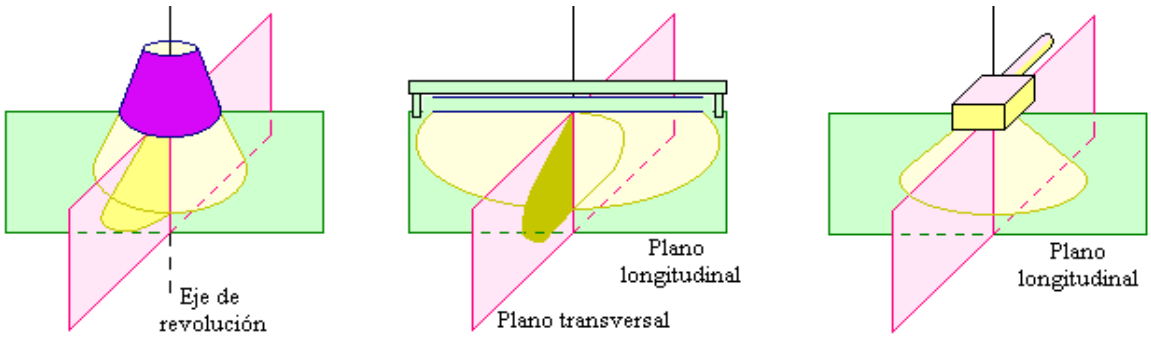


Ilustración 50: Luminarias con infinitos planos de simetría (izq.), con dos planos de simetría (centro) y con un plano de simetría (der.) [WEB08].

También es posible encontrar los datos de distribución luminosa en tablas llamadas matrices de intensidades luminosas donde para cada pareja de valores de C y γ se obtiene un valor de I_v normalizado para una lámpara de flujo de 1000 lm.

Tabla 5: Formato de la Matriz de Intensidades

	C_j
γ_i	I_{ij}

La lectura de la curva de distribución luminosa permitirá optar por la luminaria más adecuada y lograr un proyecto más económico. Una luminaria de distribución “ancha” y buen rendimiento permitirá un gran distanciamiento entre las mismas sin sacrificar la uniformidad de la iluminación.

1.9.2.2 Diagramas isocandela

Consiste en imaginar que la luminaria está en el centro de una esfera en cuya superficie exterior se unen los puntos de igual intensidad por una línea. Las superficies iguales en este diagrama representan ángulos sólidos. Por esta

razón el diagrama puede ser utilizado para calcular el flujo luminoso para una zona dada, multiplicando el área por la intensidad luminosa (teniendo en cuenta la escala a la que está representada el diagrama). Si la luminaria está instalada con un ángulo de inclinación δ, los trazos tiene que ser girados alrededor del centro en un ángulo δ para deducir las nuevas coordenadas C-γ [WEB10].

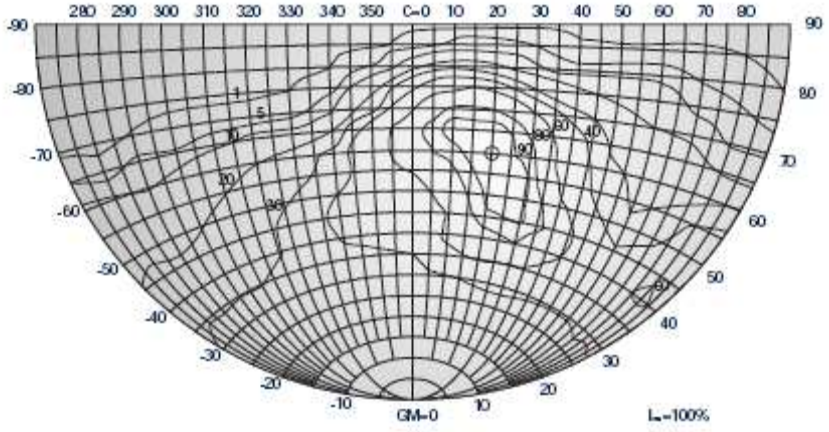


Ilustración 51: Diagrama isocandela en proyeccion azimutal [WEB10].

1.9.2.3 Diagramas isolux

Las curvas revisadas anteriormente (diagramas polares e isocandelas) se obtienen a partir de características de la fuente luminosa, flujo o intensidad luminosa, y dan información sobre la forma y magnitud de la emisión luminosa de esta. En contraste, las curvas isolux hacen referencia a las iluminancias, flujo luminoso recibido por una superficie, datos que se obtienen experimentalmente o por cálculo a partir de la matriz de intensidades usando la fórmula:

$$E_H = \frac{I(C,\gamma)}{H^2}\cos^3\gamma$$

Estos gráficos son muy útiles porque dan información sobre la cantidad de luz recibida en cada punto de la superficie de trabajo y son utilizadas especialmente en el alumbrado público donde de un vistazo nos podemos hacer una idea de como iluminan las farolas la calle. Lo más habitual es expresar las curvas isolux en valores absolutos definidos para una lámpara de 1000 lm y una altura de montaje de 1 m [WEB08].

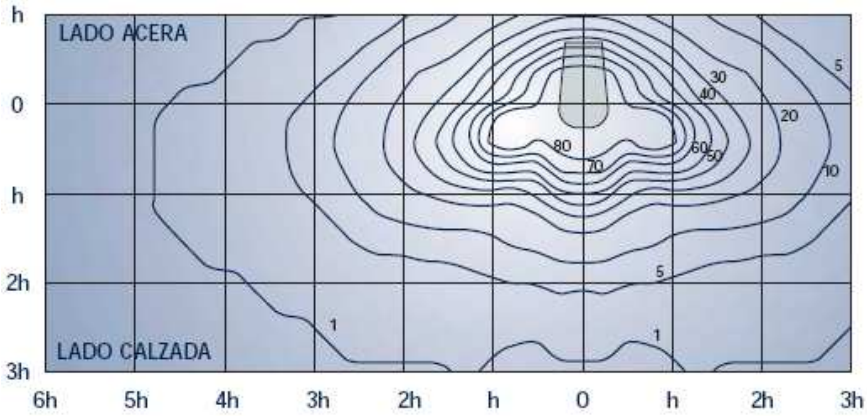


Ilustración 52: Diagrama isolux para la superficie a iluminar [WEB10]

1.9.3 Clasificación

La Ilustración 53 muestra la clasificación de las fuentes luminosas artificiales más relevantes.

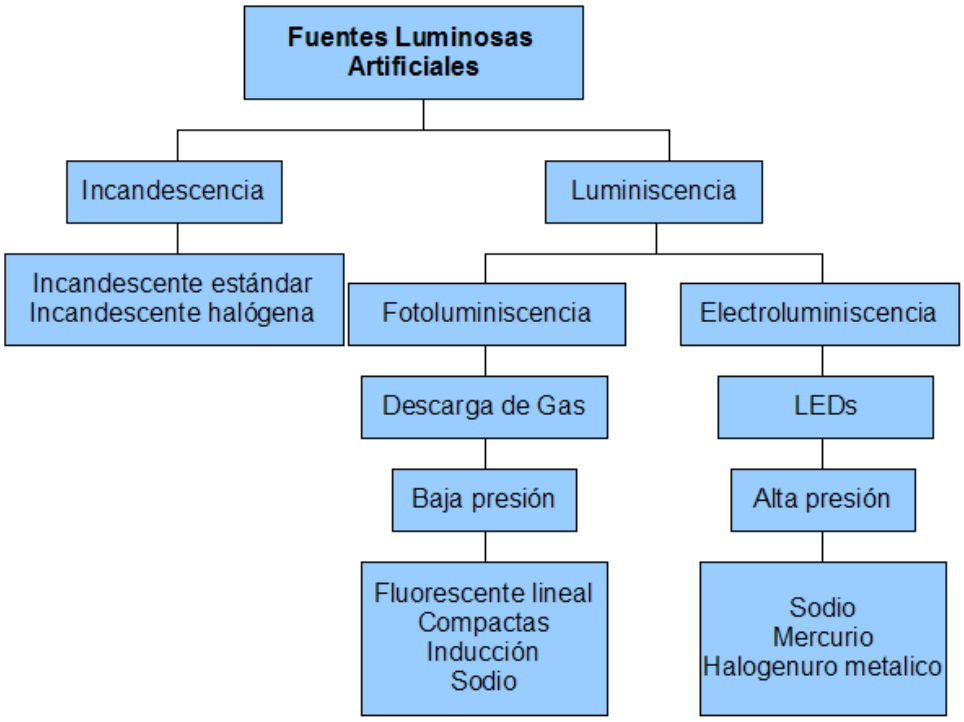


Ilustración 53: Clasificación de las fuentes luminosas más importantes [ELIA02]

1.9.4 Características Generales

Las características generales de las fuentes luminosas se pueden dividir en cuatro grupos:

1.9.4.1 Fotométricas:

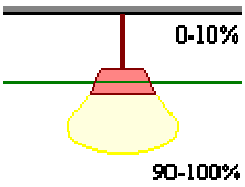
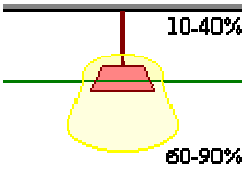
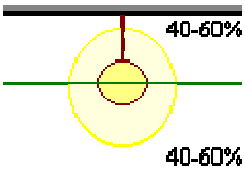
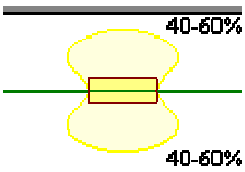
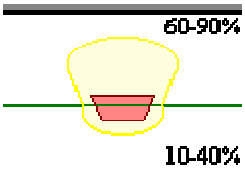
Dentro de este grupo se encuentra al flujo luminoso, la eficacia y la intensidad luminosa (véase 1.4 Magnitudes y Unidades Fotométricas). La eficiencia luminosa depende del porcentaje de la potencia eléctrica que se transforma en radiación visible y la distribución espectral emitida por la fuente en relación con la curva de sensibilidad espectral del sistema visual humano [ELIA02]. Esta característica es muy importante al momento de elegir una fuente de iluminación ya que su eficiencia influirá en el resultado de la iluminación en general y los costos relacionados con el consumo eléctrico.

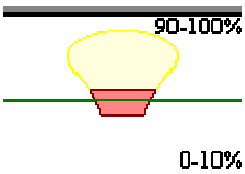
Tabla 6: Distribución de la energía emitida en la radiación de distintas fuentes luminosas [CEI96]

Tipo de Fuente	% de radiación visible	% de radiación UV	% de radiación IR	Conducción y convección
Incandescente	5.75	0.25	75	19
Fluorescente	28	0.5		71.5
Mercurio halogenado	24	1.5	24.5	50
Mercurio de alta presión	16.5	4	15	64.5
Sodio de baja presión	31		25	44
Sodio de alta presión	40.5		3.5	56

La C.I.E. clasifica las luminarias (véase Tabla 7) de acuerdo con el porcentaje de flujo emitido por el artefacto hacia cada hemisferio.

Tabla 7: Clasificación CIE de luminarias de acuerdo con la distribución del flujo luminoso hacia los hemisferios inferior y superior [ELIA02].

Tipo de Luminaria	Distribución del flujo por hemisferios $\frac{\%superior}{\%inferior}$	Características
Directa		Alta eficiencia energética. Posibilita buena uniformidad y balance de claridades en el campo visual. Con distribución concentrada puede requerir alumbrado suplementario para aumentar la iluminancia en superficies verticales. El cielorraso o cavidad dobre el plano de montaje pueden resultar poco iluminados. En general requiere control de luminancias para minimizar deslumbramiento (directo y reflejado).
Semi-directa		Similares al tipo directo pero con menor eficiencia energética. Reduce el contraste de luminarias con el cielorraso. La luz reflejada (difusa) suaviza sombras y mejora las relaciones de claridad. No deben instalarse demasiado cerca del cielorraso para evitar áreas de alta luminancia que podrían resultar distractivas, perturbadoras y afectar la estética del ambiente.
Difusa		Combinadas entre tipos directa y semi-directa pero con menor eficiencia energética. Produce buenas relaciones de claridad y suavizado de sombras. Puede ocasionar deslumbramiento (directo y reflejado) aunque su efecto es compensado por la componente reflejada (difusa). Requiere altas reflectancias de paredes y cielorraso.
Directa-indirecta		Es un caso especial del tipo difusa pero con eficiencia energética un poco mayor. Estas luminarias emiten poco flujo en ángulos próximos a la horizontal lo cual reduce las luminancias en la zona de deslumbramiento.
Semi-indirecta		Similares al tipo semi-directo pero con menor eficiencia energética. Las superficies del local deben tener alta reflectancia. La baja componente directa reduce las luminancias deslumbrantes y el contraste de claridades con el cielorraso.

Indirecta		Elimina virtualmente las sombras y el deslumbramiento directo y reflejado pero tiene baja eficiencia energética. Requiere altas reflectancias de paredes y cielorraso y un adecuado programa de mantenimiento de artefactos y superficies Hay que cuidar el balance de luminancias con el cielorraso.

1.9.4.2 Colorimétricas

Dentro de las características colorimétricas tenemos a la Temperatura del Color (Tc) y al Índice de Rendimiento de Color (IRC), dichas magnitudes fueron desarrolladas y explicadas en 1.4 Magnitudes y Unidades Fotométricas.

1.9.4.3 Eléctricas

Una de las diferencias fundamentales entre las lámparas incandescentes y las de descarga es que las primeras tienen una resistencia eléctrica positiva y en las de descarga ocurre lo contrario, debido a la compensación requerida por estas lámparas se justifica el uso de balastos para su funcionamiento. Dentro de las características eléctricas tenemos que considerar el arranque, periodo de encendido y el reencendido [ELIA02].

1.9.4.4 Duración

Dentro de la duración de una fuente luminosa se destaca la vida de esta, el cual depende de un sinnúmero de factores como por ejemplo el número de encendidos, la posición de funcionamiento, la tensión de la fuente de alimentación y otros factores ambientales como temperatura y vibraciones. Según la ELI de Argentina [ELIA02], las diferentes formas de definir la vida son:

- **Vida individual**: es el número de horas de encendido después del cual una lámpara queda inservible, bajo condiciones específicas.

- **Vida promedio o nominal**: tiempo transcurrido hasta que falla el 50% de las lámparas de la muestra bajo condiciones específicas.

- **Vida útil o económica**: valor basado en datos de depreciación, cambio de color, supervivencia como así también el costo de la lámpara, precio de energía que consume y costo de mantenimiento. Puede definirse como el número de horas durante el cual puede operar correctamente una lámpara hasta que se hace necesario su reemplazo.

- **Vida media**: valor medio estadístico sobre la base de una muestra.

Otra característica tomada en cuenta para definir la duración de una lámpara según la [ELIA02] es la Depreciación del Flujo Luminoso, el cual corresponde al valor medido luego de 100 horas de funcionamiento y cuyo calor puede disminuir como consecuencia del ennegrecimiento del bulbo o agotamiento gradual.

La Tabla 8 compara la vida nominal de diferentes fuentes de luz y el porcentaje de depreciación luminosa al 50% y 100% de su vida nominal.

Tabla 8: Vida nominal y depreciación luminosa para distintos tipos de lámparas [NAR00].

Fuente de luz	Vida nominal (h)	% depreciación luminosa al 50% de la vida nominal (lm)	% depreciación luminosa al 100% de la vida nominal (lm)
Incandescente	1000	88	83
Incandescente halogenada	2000	98	97
Fluorescente T8	20000	85	75
Mercurio	24000	75	65
Mercurio halogenado	15000	74	68
Sodio de alta presión	24000	90	72

La Tabla 9 muestra las características fotométricas, colorimétricas y de duración para las lámparas mas representativas de cada tipo.

Tabla 9: Características fotométricas, colorimétricas y de duración para las lámparas mas representativas de cada tipo.

Lámpara	Potencia (W)	Temperatura de color (K)	Eficacia (lm/W)	IRC	Vida útil (h)	Tiempo de encendido (min)
Incandescente convencional	100	2700	15	100	1000	0
Inc. Halógena lineal	300	2950	18	100	2000	0
Inc. Halógena reflectora	100	2850	15	100	2500	0
Inc. Halógena de baja tensión	50	3000 – 3200	18	100	3000	0
Fluorescente lineal T5 alta frecuencia	28	3000 - 4100	104	85	12000	0
Fluorescente lineal T8 alta frecuencia	32	3000 – 4100	75	85	12000	0
Fluorescente compacta	36	2700 – 4100	80	85	12000	0 – 1
Fluorescente compacta doble	26	2700 – 4100	70	85	12000	0 – 1
Vapor de mercurio	125	6500	50	45	16000	< 10
Mercurio halogenado (baja potencia)	100	3200	80	75	12000	< 5
Mercurio halogenado (alta potencia)	400	4000	85	85	16000	< 10
Sodio de alta presión (baja potencia)	70	2100	90	21	16000	< 5
Sodio de alta presión (alta potencia)	250	2100	104	21	16000	< 5

1.9.5 Criterios de Selección

Las características fotométricas, cromáticas, eléctricas y de duración junto al programa de actividades y objetivos del espacio a iluminar, así como las consideraciones arquitectónicas y económicas constituyen los condicionantes a la hora de elegir las fuentes luminosas.

Según la ELI [ELIA02], los criterios de selección de luminarias de pueden dividir en:

1.9.5.1 Criterios de Eficacia

Como se planteo anteriormente, la característica fotométrica más importante en cuanto a las consideraciones de energía es la eficacia, ya que cuanto mas eficiente es una lámpara, menos energía necesita para producir la misma cantidad de luz. Por ejemplo, las lámparas de sodio de baja presión (SBP) tienen el mayor valor de eficacia hasta de 200 lm/W, luego le siguen las lámparas de sodio de alta presión (SAP) con un valor de hasta 130 lm/W y luego las fluorescentes lineales con 100 lm/W.

Otros criterios también deben ser considerados antes de elegir una lámpara por su eficacia, por ejemplo, las lámparas SBP generan una luz monocromática sin capacidad de reproducción de colores o índice de reproducción cromática (IRC). Cuando se requiere una lámpara con una buena reproducción de colores, las lámparas fluorescentes compactas (LFC) son una opción interesante, y son utilizadas con mayor frecuencia en viviendas y ciertas aplicaciones comerciales. Y cuando se requiere alta eficacia (hasta 100 lm/W) y excelente reproducción de colores con alto flujo luminoso, las lámparas de halogenuros metálicos son ideales [ELIA02].

1.9.5.2 Criterios Cromáticos

Desde el punto de vista de la reproducción cromática de las fuentes que presentan el mayor índice son las incandescentes halógenas y convencionales (IRC = 100), pero por otra parte estas tienen baja eficacia. Si la iluminación de

interiores y de exteriores donde el IRC es tan importante como la eficacia, las lámparas fluorescentes trifosforadas y de mercurio halogenado son una excelente opción, ya que tienen un valor de IRC de aproximadamente 80 y una eficacia muy superior a las lámparas incandescentes.

1.9.5.3 Criterios de Duración

Las lámparas con el mayor tiempo de vida son las de inducción, luego las de mercurio de alta presión y sodio de alta presión (16000 a 24000 horas véase Tabla 8). El grupo de lámparas incandescentes constituyen el grupo más desfavorable, con una vida nominal de 1000 horas.

1.10 Métodos de Iluminación

1.10.1 Introducción

Dentro de las etapas del diseño de la iluminación de un lugar de trabajo se encuentra la elección del sistema de alumbrado o método de iluminación. Esto se logra luego de un análisis de las características luminosas necesarias para satisfacer las demandas que el trabajo plantea.

Dependiendo del grado de uniformidad de la iluminación, los métodos de iluminación o alumbrado se dividen en los siguientes:

1.10.2 Alumbrado General

Este tipo de alumbrado se caracteriza por proveer una iluminación uniforme en toda el área del local ya que las luminarias se distribuyen de una manera regular (Ilustración 54). Este método de alumbrado influye en un consumo de energía mayor por alumbrado, en especial en locales de planta abierta. El alumbrado general es simple de diseñar y no requiere coordinación con el esquema de distribución de los puestos de trabajo.

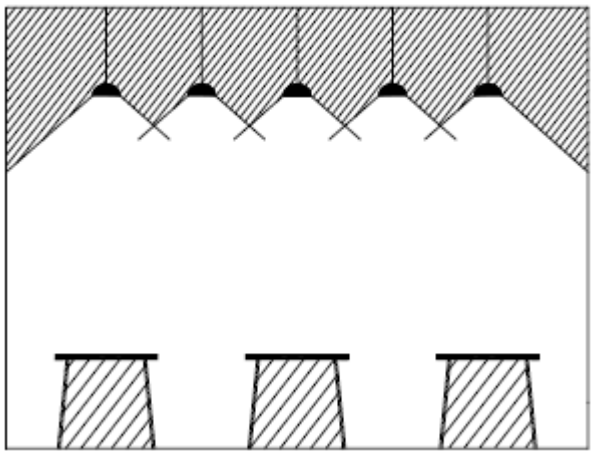


Ilustración 54: Alumbrado general [ELIA02]

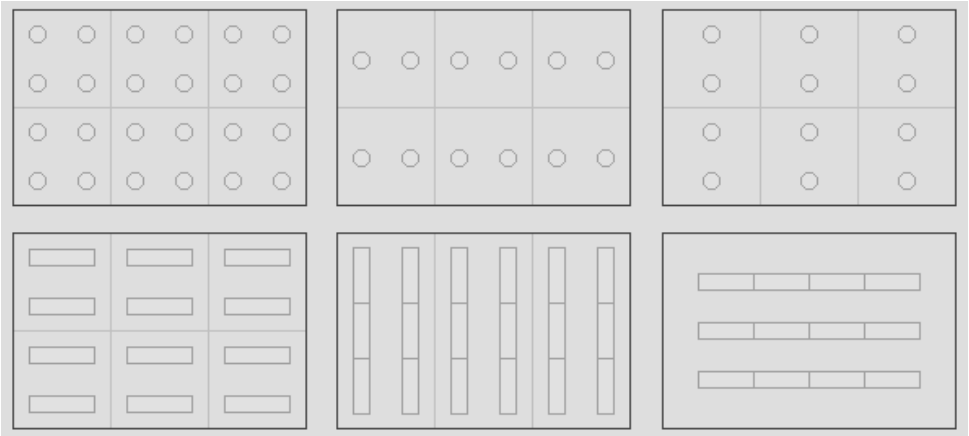


Ilustración 55: Ejemplos de distribución de luminarias en alumbrado general [ELIA02]

1.10.3 Alumbrado Localizado

El alumbrado localizado se distribuye de manera que los valores de iluminación altos sean exclusivos para las áreas de trabajo o sectores destacados (Ilustración 56) como accesos, áreas con riesgo de accidentes, efectos decorativos, etc. y se deja al resto de la instalación con niveles menores. De esta manera, el consumo eléctrico se reduce de manera considerable en comparación con el método de alumbrado general, pero el diseño se torna más complejo al considerar muchos más aspectos como por ejemplo el evitar efectos de distracción y el cansancio o fatiga visual.

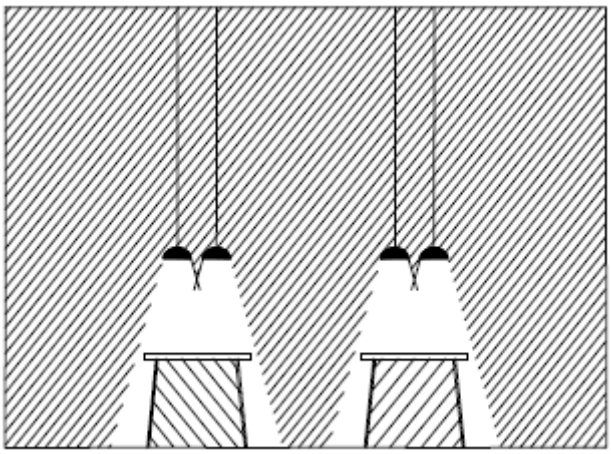


Ilustración 56: Alumbrado localizado [ELIA02].

1.10.4 Alumbrado General Localizado

El alumbrado general localizado combina características del alumbrado general y el localizado (Ilustración 57), de manera que los niveles de iluminación mas altos se enfocan donde se realizan las tareas y los niveles mas bajos o generales se distribuyen de manera uniforme por todo el local.

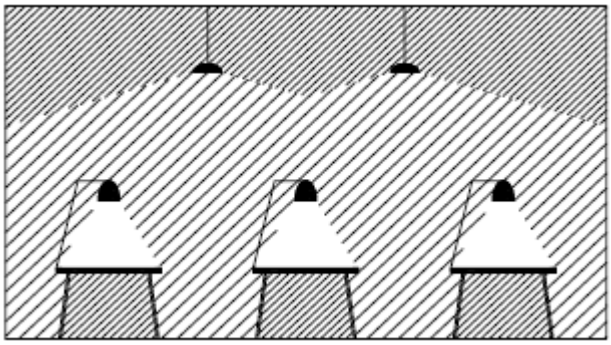


Ilustración 57: Alumbrado general localizado [ELIA02].

1.10.5 Alumbrado Modularizado

El alumbrado modularizado brinda una iluminación uniforme por sectores o módulos de las áreas de trabajo. Este tipo de alumbrado requiere de un diseño más exigente debido a la consideración de factores adicionales sobre el ambiente laboral a iluminar. Este método además requiere un consumo de energía elevado ya que se necesita de sectorización de los circuitos.

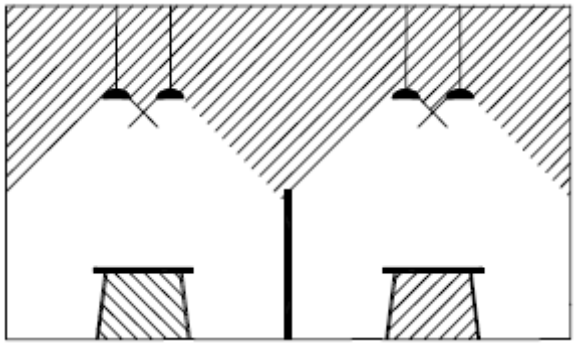


Ilustración 58: Alumbrado modularizado [ELIA02].

Tabla 10: Caracteristicas aproximadas de los sistemas de iluminacion [ELIA02]

Sistema de Alumbrado	Disposición de Luminarias	Características Luminotécnicas	Efectos Visuales		Coordinación con ubicación de áreas de trabajo	Consumo energético
			Sobre el Espacio	Sobre personas y objetos		
General Directo o indirecto	Uniforme	Altos niveles de Iluminancia en todo el espacio. Excelente uniformidad. Reducción de contrastes y brillos. Se minimiza la proyección de sombras.	Produce sensación de amplitud y orden Crea atmósferas de monotonía y condiciones propicias pa-ra trabajos que requieren de alta concentración.	Modelados blandos. Aplana texturas. Oculta detalles. Minimiza efectos de reflejos especulares Apaga intensidad de los colores.	No requiere	Elevado (más con sistema indirecto). No permite reducción individual de los niveles de iluminación.
Localizado	Irregular	Altos niveles de Iluminancia sólo en áreas de interés. Uniformidad general baja Contrastes realzados. Puede cau-sar importante proyec-ción de sombras	Produce sensación de reducción del espacio. Puede crear atmósferas dramáticas, estimulantes y distractivas	Modelados duros. Realza textura y detalles. Los colores resultan más inten-sos. Ideal para crear efectos luminosos.	Muy importante	Reducido. Adecuado para controlar niveles de iluminación individualmente.
General y localizado	Uniforme (general) e irregular (localizado)	Iluminancia general reducida respecto de áreas de trabajo. Unifor-midad general baja. Contrastes realzados. Puede causar impor-tante proyección de sombras	Un balance adecua- do puede compen-sar la sensación de reducción del espa-cio y crear condicio-nes propicias para el trabajo	Con un balance adecuado el modelado resulta casi natural. Buena apariencia de textura y detalles.	Muy importante sólo para el sistema de alumbrado localizado	Intermedio entre alumbrado general y localizado. Adecuado para controlar niveles de iluminación individualmente sin afectar el resto de la instalación.
Modularizado	Uniforme por sectores	Iluminancia media elevada. Uniformidad excelente. Reducidos contrastes y proyección de sombras	Idem a alumbrado general	Idem a alumbrado general	Importante para determinar el arreglo de luminarias	Elevado. Requiere sectorización de los circuitos. Permite reducción de los nive-les de iluminación por sectores.

1.11 Métodos de Cálculo de Iluminación de Interiores

1.11.1 Método de las Cavidades Zonales

El Método de las Cavidades Zonales, llamado también Método de los Lúmenes, es el método desarrollado por los investigadores J. R. Jones y B. F. Jones y recomendado por la I.E.S. (Illuminating Engineering Society – U.S.A.) [DESC99]. Este método es el recomendado por la A.A.D.L., Asociación Argentina de Luminotecnia según la norma IRAM-AADL j20-05 sobre alumbrado de interiores [ELIA02].

La finalidad del método de las cavidades zonales es calcular el valor medio en servicio de la iluminancia en un local iluminado con alumbrado general. Es muy práctico y fácil de usar, y por ello se utiliza mucho en la iluminación de interiores cuando la precisión necesaria no es muy alta como ocurre en la mayoría de los casos [WEB08].

Este método permite considerar:

- Altura variable de suspensión de las luminarias
- Altura variable del plano de trabajo
- Distintas reflectancias de paredes sobre y bajo el plano de trabajo y por arriba del plano de las luminarias.
- Obstrucciones en el espacio existente sobre el plano de las luminarias (por ejemplo vigas).
- Planta del local compuesta por mas de un rectángulo.

Este método, como su nombre sugiere, divide al local en tres cavidades: la cavidad de cielorraso (ceiling cavity), la cavidad del local (room cavity) y la cavidad de piso (floor cavity). Esta forma de analizar por separado el comportamiento de los tres sectores más importantes de un local a iluminar, confiere a los cálculos realizados por este método una mayor precisión.

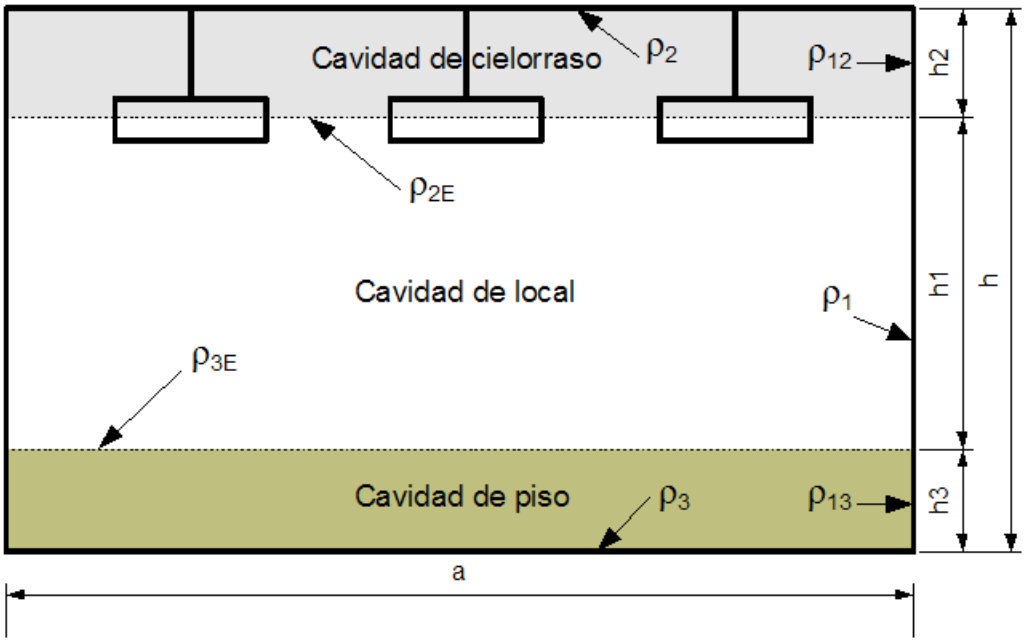


Ilustración 59: División de un local en cavidades.

1.11.1.1 Índice de cavidad

Cada cavidad queda caracterizada por un índice denominado k que depende del tipo de local. Para locales rectangulares la formula queda expresada por:

$$k = \frac{5h(a+1)}{al}$$

donde:

h = altura del local
a = ancho del local
l = largo o profundidad del local

Para espacios irregulares, la formula queda definida como:

$$k = \frac{2.5h(perimetro)}{al}$$

Y para espacios circulares:

$$k = \frac{10h}{diametro}$$

Según [DESC99], se reemplaza el complejo análisis de la distribución del flujo luminoso emitido por las luminarias y sus interreflexiones arriba de las mismas y debajo del plano de trabajo, por las reflexiones en los planos aparentes de la luminaria y de trabajo (véase Ilustración 59) a los cuales se les asigna reflectancias efectivas ρ_{2E} y ρ_{3E} que tienen en cuenta las reflectancias reales que limitan las cavidades de cielorraso y piso, cuyas dimensiones quedan caracterizadas por los índices k_2 y k_3 respectivamente.

$$k_1 = \frac{5h_1(a+1)}{a.l}$$ índice de cavidad de local

$$k_2 = \frac{5h_2(a+1)}{a.l} = k_1 \frac{h_2}{h_1}$$ índice de cavidad de cielorraso

$$k_3 = \frac{5h_3(a+1)}{a.l} = k_1 \frac{h_3}{h_1}$$ índice de cavidad de piso

Si $h_2 = 0$, entonces las luminarias son de tipo embutidas o a nivel de cielorraso, por lo tanto $\rho_{2E} = \rho_2$.

Si $h_3 = 0$, entonces el plano de trabajo es el piso, por lo tanto $\rho_{3E} = \rho_3$.

1.11.1.2 Coeficiente de utilización

El coeficiente de utilización η, se define como el cociente entre el flujo útil que llega al plano de trabajo y el flujo emitido por las lámparas contenidas en una cantidad N de luminarias [DESC99].

$$u = \frac{\Phi_u}{N.\Phi_L}$$

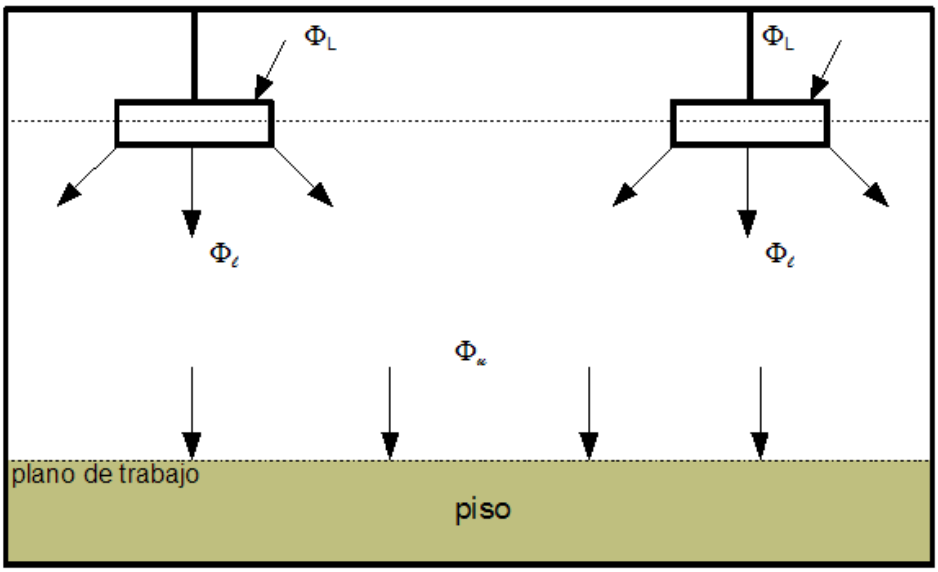


Ilustración 60: El coeficiente de utilización.

Una parte del flujo de la lámpara Φ_L es absorbido por la propia luminaria (perdidas por absorción en los materiales reflectantes y transmisores). La luminaria tiene, por lo tanto un rendimiento ε:

$$\varepsilon = \frac{\Phi_l}{\Phi_L}$$

El flujo emitido por la luminaria Φ_l, sufre a su vez una serie de interreflexiones antes de llegar al plano de trabajo y parte de ese flujo emitido es también absorbido por las paredes que limitan el local. Entonces el rendimiento luminoso dependerá de sus dimensiones y reflectancias.

$$\eta = \frac{\Phi_u}{\Phi l.N}$$

El rendimiento total considerando el flujo absorbido por la luminaria y el absorbido por las superficies del local será:

$$\varepsilon.\eta = \frac{\Phi_l}{\Phi_L}.\frac{\Phi_u}{\Phi_l.N} = u$$

Es decir que el factor de utilización representa el producto entre el rendimiento de las luminarias ε y el rendimiento del local η. Podemos escribir:

$$\Phi_u = u.N.\Phi_L$$

La iluminancia sobre el plano de trabajo es:

$$E = \frac{\Phi_u}{A_r} = \frac{u.N.\Phi_L}{a.l}[lux]$$

$$\overline{E_i} = \frac{u.N.\Phi_L}{a.l}[lux]$$

Entonces $\overline{E_i}$ representa la iluminancia media inicial sobre el plano de trabajo.

1.11.1.3 Luminancias medias

En [DESC99] y en base al Anexo IV Ley Nº 19587 se muestran las siguientes relaciones de luminancias medias:

Entre el centro de atención (campo de tarea visual) y alrededor inmediato (superficie de trabajo).	3:1
Entre centro de atención y alrededor mediato o espacio circundante.	10:1
Máxima diferencia de luminancias en el campo visual.	40:1

Si se trata de superficies asimilables o perfectamente difusoras como lo son las de terminación mate se puede utilizar la siguiente relación:

$$\overline{L} = \rho\frac{\overline{E}}{\pi}$$

Para permitir un cálculo rápido y sencillo de las luminancias, el método de las cavidades zonales introduce como complemento el concepto del coeficiente de

luminancia aplicando la relación anterior [DESC99]. Para ello es suficiente sustituir en:

$$\overline{E_i} = \frac{u.N.\Phi_L}{a.l}[lux]$$

el coeficiente de utilización u por el correspondiente coeficiente de luminancia q_1 y q_{2E} dividido por π.

Luminancia media inicial de la pared:

$$\overline{L}_{1i} = \frac{q_1.\Phi_L.N}{\pi.l.a}[cd.m^{-2}]$$

Luminancia media inicial de la superficie aparente de cavidad de cielorraso:

$$\overline{L}_{2Ei} = \frac{q_{2E}.\Phi_L.N}{\pi.l.a}[cd.m^{-2}]$$

1.11.1.4 Uniformidad de la iluminancia

En los sistemas de iluminación general, las luminarias se distribuyen uniformemente en planta como se indica en la Ilustración 61; pero también es posible adoptar otras distribuciones ya que la condición de uniformidad se refiere a la densidad de artefactos.

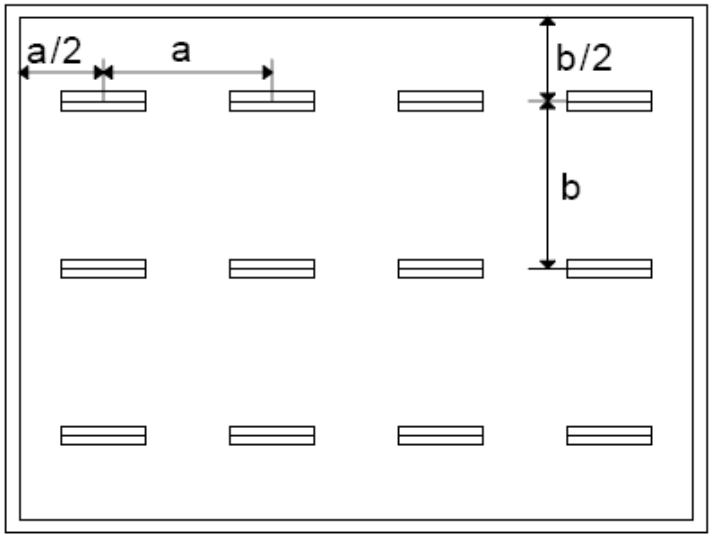


Ilustración 61: Distribución uniforme típica de luminarias [ELIA02].

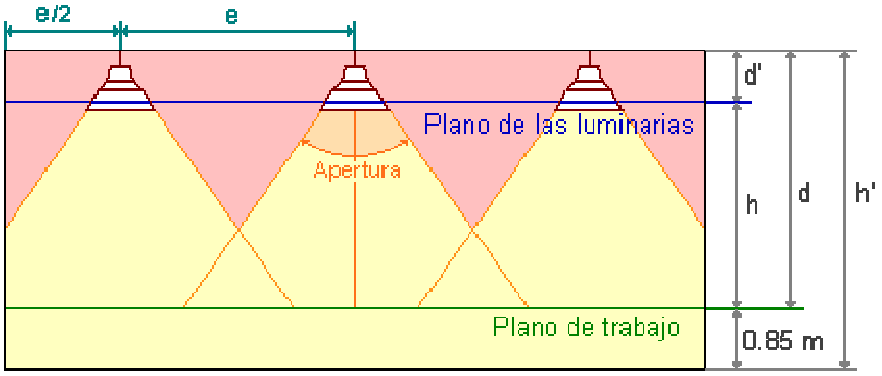


Ilustración 62: Separación de luminarias de acuerdo a su apertura del haz de luz [WEB08].

La distancia máxima de separación entre las luminarias dependerá del ángulo de apertura del haz de luz y de la altura de las luminarias sobre el plano de trabajo [EDILUM]. Como puede verse en la Ilustración 62, mientras más abierto sea el haz y mayor la altura de la luminaria más superficie iluminará aunque será menor el nivel de iluminancia que llegará al plano de trabajo tal y como dice la ley inversa de los cuadrados [DESC99]. De la misma manera, se puede apreciar que las luminarias próximas a la pared necesitan estar más cerca para iluminarla (normalmente la mitad de la distancia). Las conclusiones sobre la separación entre las luminarias las podemos resumir como sigue:

Tabla 11: Criterios de separación entre luminarias [WEB08].

Tipo de luminaria	Altura del local	Distancia máxima entre luminarias
intensiva	menor a 10 m	e ≤ 1.2 h
extensiva	6 m a 10 m	e ≤ 1.5 h
semi-extensiva	4 m a 6 m	e ≤ 1.5 h
extensiva	mayor o igual a 4 m	e ≤ 1.6 h
distancia pared-luminaria: e/2		

Si después de calcular la posición de las luminarias nos encontramos que la distancia de separación es mayor que la distancia máxima admitida quiere decir que la distribución luminosa obtenida no es del todo uniforme. Esto puede deberse a que la potencia de las lámparas escogida sea excesiva. En estos casos conviene rehacer los cálculos probando a usar lámparas menos potentes, más luminarias o emplear luminarias con menos lámparas [WEB08].

1.11.1.5 Factor de mantenimiento y factor de compensación

El factor de mantenimiento (fm) contempla el nivel de ensuciamiento del local. Este factor es muy significativo dentro de la depreciación del sistema de iluminación. La E.L.I. Argentina [ELIA02] recomienda un buen plan y estrategias de mantenimiento y reemplazo de lámparas.

En la Ilustración 63, la curva A indica la reducción de iluminancia si solo actuara la depreciación de lámparas y la curva C la variación real como resultado del mantenimiento. Cuando se efectúa la limpieza de luminarias únicamente (por ejemplo al final de los años 1 y 2) no se reestablece hasta el nivel dado por la curva A, ya que actúa también la depreciación del local (curva B).

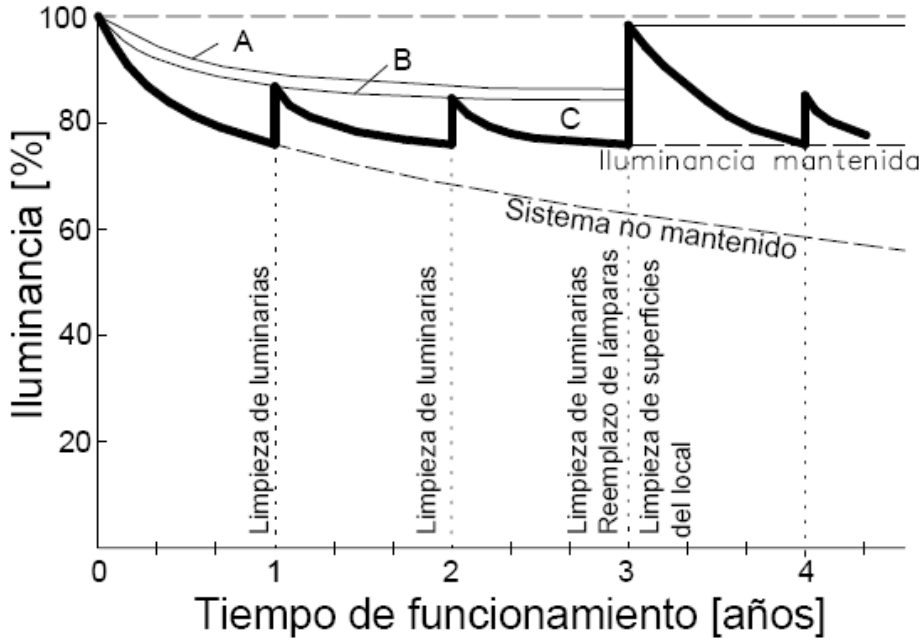


Ilustración 63: Esquema de mantenimiento de una instalación de iluminación [ELIA02].

Tabla 12: Valores recomendados por [DESC99] para los factores de mantenimiento y de compensación.

Nivel de ensuciamiento	Factor de mantenimiento (ensuciamiento)		Factor de compensación K_c = 1/fac. mant.
	Local	Lámpara	
Bajo	0.90	0.90	1.25
Mediano	0.80	0.90	1.40
Alto	0.70	0.90	1.60

1.11.1.6 Reflectancias medias

Según [DESC99], la reflectancia media queda definida por la sumatoria del producto entre las reflectancias y el área de sus respectivas superficies:

$$\overline{\rho} = \frac{\sum_{i=1}^{i=n} \rho_i . S_i}{\sum_{i=1}^{i=n} S_i} 100$$

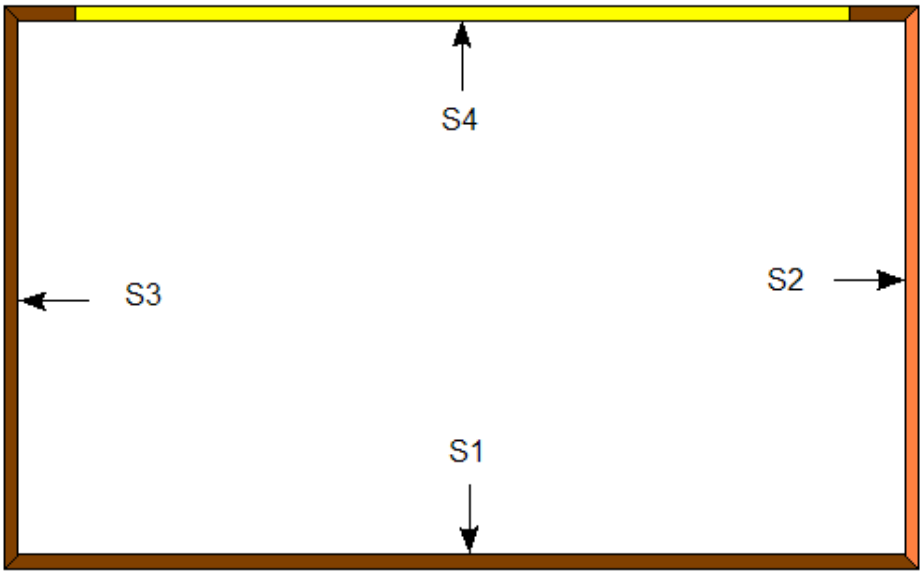


Ilustración 64: Superficies de distintos materiales en un local [DESC99].

$$\rho = \frac{\rho_1 S_1 + \rho_2 S_2 + \rho_3 S_3 + \rho_4 S_4}{S_1 + S_2 + S_3 + S_4} 100$$

Por ejemplo para los valores de la Tabla 13:

Tabla 13: Factores de reflexión ρ y superficies S de diferentes tipos de materiales [DESC99].

Madera oscura	ρ_1 = 10%	S_1 = 6 m^2
Ladrillo claro	ρ_2 = 30%	S_2 = 60 m^2
Madera oscura	ρ_3 = 30%	S_3 = 18 m^2
Superficie vidriada	ρ_4 = 0%	S_4 = 16 m^2

La reflectancia media se calcula de la siguiente forma:

$$\rho = \frac{0.10(6) + 0.30(60) + 0.10(18) + 0(16)}{6 + 60 + 18 + 16} 100 = 20\%$$

1.11.1.7 Ejemplo Práctico del Método de las Cavidades Zonales

Un auditorio típico tiene 60 pies de largo y 30 pies de ancho con una altura del techo de 14 pies. Las reflectancias son: techo 80%, paredes 30%, piso 10%. Cuatro lámparas Prismawrap de 3150 lm (coeficientes de utilización mostrados en la Tabla 14) se utilizaran colgadas a 4 pies, y el plano de trabajo es de 2 pies sobre el piso. Encontrar el nivel de iluminación si hay 18 luminarias en el local.

Solución:

(1) Calcular los índices de cavidad (cavity ratios) como se muestra a continuación:

$$RCR = k1 = \frac{5(8)(30 + 60)}{30(60)} = 2.0$$

$$CCR = k2 = \frac{5(4)(30 + 60)}{30(60)} = 1.0$$

$$FCR = k3 = \frac{5(2)(30 + 60)}{30(60)} = 0.5$$

(2) En la
Tabla 15, buscar las reflectancias efectivas de cavidad para estas cavidades de techo y piso. pcc para la cavidad de techo (ceiling cavity) es determinada como 62%, mientras que pfc para la cavidad de piso (floor cavity) es 10%.

(3) Al conocer el índice de cavidad del local (room cavity ratio RCR), es posible encontrar el coeficiente de utilización para la luminaria Prismawrap en una habitación que tiene un RCR de 2.0 y reflectancias efectivas de :

pcc = 62%; pw = 30%; pfc = 20%.

Por interpolación entre los valores de la tabla (.56 y .54), el CU es .55. Nótese que este CU es para una reflectancia efectiva de 20% mientras que la reflectancia efectiva del piso pfc es 10%. Para corregir esto, se localiza el apropiado multiplicador en la Tabla 16 para el RCR ya calculado de 2.0. Resulta entonces .962 y es encontrado al interpolar entre los números .957 y .968 de la Tabla 16 para pcc de 70%, pw de 30% y pcc de 50%, pw de 30% con un RCR de 2.0.

Entonces:

CU final = .55 x .962 = .53

(4) El nivel de iluminación puede ahora ser calculado si conocemos el numero de unidades que se van a utilizar y los lúmenes de la misma.

$$\overline{E} = \frac{u.N.\Phi_L}{a.l}[lux]$$

$$\overline{E} = \frac{0.53(18)(3150)(4)}{60(30)}$$

$$\overline{E} = 67\,\text{lux}$$

Una posible distribución de las lámparas es 3 columnas de 6 lámparas separadas 10 pies en cada dirección. El criterio de separación es 1.4, lo que indica una separación máxima de 11.2 pies. El espaciado actual es menor que el máximo permitido. Entonces, la iluminación sobre el plano de trabajo debería ser uniforme.

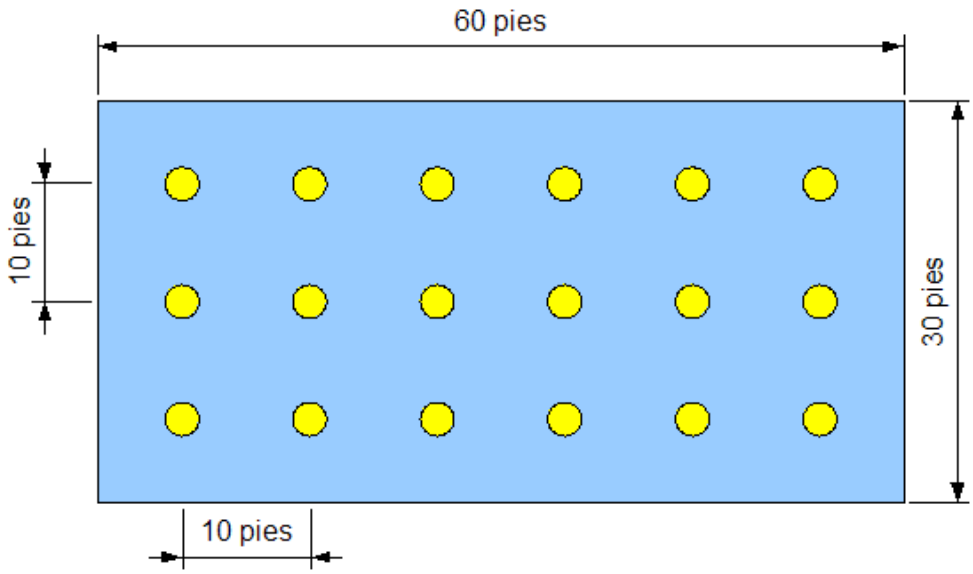


Ilustración 65: Distribución final de las luminarias.

Tabla 14: Luminaria Prismawrap de 4 lámparas – Coeficientes de Utilización

pcc		80%				70%				50%		
pw		70%	50%	30%	10%	70%	50%	30%	10%	50%	30%	10%
0		.78	.78	.78	.78	.75	.75	.75	.75	.70	.70	.70
1		.72	.69	.67	.64	.69	.67	.65	.63	.63	.61	.59
2		.66	.62	.58	.55	.64	.60	.56	.53	.56	.54	.51
3		.61	.55	.51	.47	.59	.54	.50	.46	.51	.47	.44
4	R	.57	.50	.45	.41	.55	.48	.44	.40	.46	.42	.39
5	C	.52	.45	.39	.35	.50	.43	.38	.35	.41	.37	.34
6	R	.48	.40	.35	.31	.47	.39	.34	.31	.37	.33	.30
7		.45	.36	.31	.27	.43	.35	.30	.27	.34	.29	.26
8		.41	.33	.27	.23	.40	.32	.27	.23	.30	.26	.23
9		.38	.29	.24	.20	.36	.28	.23	.20	.27	.23	.20
10		.35	.26	.21	.18	.34	.26	.21	.18	.25	.20	.17
sep	1,4											

Tabla 15: Porcentajes de reflectancias efectivas de cavidad de cielorraso o suelo.

% ceiling or floor reflectance	80%				50%			10%		
% wall reflectance	80%	70%	50%	30%	70%	50%	30%	50%	30%	10%
Cavity ratio										
0.2	78	78	77	76	49	48	47	10	10	9
0.4	77	76	74	72	48	47	45	11	10	9

0.6	76	75	71	68	47	45	43	11	**10**	8
0.8	75	73	69	65	47	44	40	11	10	8
1.0	74	72	67	**62**	46	43	38	12	10	8
1.2	73	70	64	58	45	41	36	12	10	7
1.4	72	68	62	55	45	40	35	12	10	7
1.6	71	67	60	53	44	39	33	12	9	7
1.8	70	66	58	50	43	38	31	13	9	6
2.0	69	64	56	48	43	37	30	13	9	6

Tabla 16: Factores de corrección para porcentajes de reflectancia efectiva de cavidad de piso diferentes a 20%

pcc	**80%**				**70%**				**50%**		
pw	**70%**	**50%**	**30%**	**10%**	**70%**	**50%**	**30%**	**10%**	**50%**	**30%**	**10%**
For 30 per cent effective floor cavity reflectance (20 per cent = 1.00)											
RCR											
1	1.092	1.082	1.075	1.068	1.077	1.070	1.064	1.059	1.049	1.044	1.040
2	1.079	1.066	1.055	1.047	1.068	1.057	1.048	1.039	1.041	1.033	1.027
3	1.070	1.054	1,042	1.033	1.061	1.048	1.037	1.028	1.034	1.027	1.020
4	1.062	1.045	1.033	1.024	1.055	1.040	1.029	1.021	1.030	1.022	1.015
5	1.056	1,038	1.026	1.018	1.050	1.034	1.024	1.015	1.027	1.018	1.012
6	1.052	1.033	1.021	1.014	1.047	1.030	1.020	1.012	1.024	1.015	l.009
7	1.047	1.029	1.018	1.011	1.043	1.026	1.017	l.009	1.022	1.013	1.007
8	1.044	1.026	1.015	1.009	1.040	1.024	1.015	1.007	1.020	1.012	1.006
9	1.040	1.024	1.014	1.007	1.037	1.022	1.014	1.006	1.019	1.011	1.005
10	1.037	1.022	1.012	1.006	1.034	1.020	1.012	1.005	1.017	1.010	1.004
For 10 per cent effective floor cavity reflectance (20 per cent=1.00)											
RCR											
1	.923	.929	.935	.940	.933	.939	.943	.948	.956	.960	.963
2	.931	.942	.950	.958	.940	.949	**.957**	.963	.962	**.968**	.974
3	.939	.951	.961	.969	.945	.957	.966	.973	.967	.975	.981
4	.944	.958	.969	.978	.950	.963	.973	.980	.972	.980	.986
5	.949	.964	.976	.983	.954	.968	.978	.985	.975	.983	.989
6	.953	.969	.980	.986	.958	.972	.982	.989	.977	.985	.992
7	.957	.973	.983	.991	.961	.975	.985	.991	.979	.987	.994
8	.960	.976	.986	.993	.963	.977	.987	.993	.981	.988	.995
9	.963	.978	.987	.994	.965	.979	.989	.994	.983	.990	.996
10	.965	.980	.965	.980	.967	.981	.990	.995	.984	.991	.997

CAPITULO II

GRÁFICOS TRIDIMENSIONALES CON OPENGL

2 GRÁFICOS TRIDIMENSIONALES CON OPENGL

2.1 Una breve historia sobre los Gráficos en Computadora

Las primeras computadoras consistían en filas y filas de interruptores y luces, en las cuales técnicos e ingenieros trabajaban por horas, días, semanas para programarlas y leer los resultados de sus cálculos. Patrones de bulbos iluminados o impresiones muy básicas brindaban información útil a los usuarios.

Los tiempos habían cambiado, luego, los datos eran almacenados eficientemente en cintas magnéticas, en discos duros, inclusive en filas de orificios en tarjetas de papel. El "hobby" de los gráficos por computadora nació el día que las computadoras empezaron a imprimir. Ya que cada caracter del alfabeto tenia una forma y tamaño definida, programadores creativos en los 70's creaban patrones e imágenes artísticas hechas por nada más que letras del abecedario.

2.1.1 Llega el CRT

El papel como un medio de salida para las computadoras es útil y actualmente aún persiste. Las impresoras láser y las de inyección de tinta han reemplazado al arte ASCII por representaciones artísticas en calidad fotográfica. El papel, sin embargo, puede ser costoso al reemplazarlo regularmente, y el usarlo constantemente es un desperdicio de recursos naturales, especialmente porque la mayoría del tiempo realmente no necesitamos una copia tangible de cálculos o consultas a una base de datos.

El tubo de rayos catódicos (CRT)[16] fue una adición tremendamente útil para la computadora. Los monitores originales CRTs eran inicialmente solo terminales de video que desplegaban texto ASCII de la misma manera que las primeras terminales de impresión, pero los CRTs eran perfectamente capaces de desplegar además puntos y líneas. Los programadores usaron las

[16] El tubo de rayos catódicos, o CRT, fue desarrollado por Ferdinand Braun, un científico Alemán, en 1897

computadoras y sus monitores para crear gráficos que suplementaron la salida textual o tabular. Los primeros algoritmos para crear líneas y curvas fueron desarrollados y publicados; los gráficos por computadora se convirtieron en una ciencia más que un pasatiempo.

Los primeros gráficos mostrados en estas terminales eran bidimensionales o 2D. Estas líneas planas, círculos y polígonos fueron usados para crear gráficos para una variedad de propósitos. Grafos y plots podían mostrar datos científicos o estadísticos en una manera que las tablas y figuras no podían. Programadores más aventureros inclusive crearon juegos simples como Pong[17] usando gráficos que consistían en poco más que líneas dibujadas que se refrescaban (redibujaban) varias veces por segundo, llamados en el mundo de los videojuegos como sprites[18].

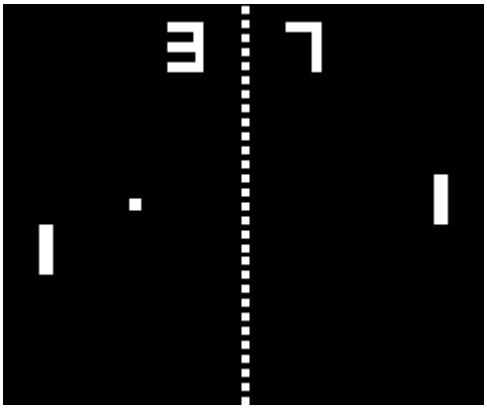


Ilustración 66: El videojuego Pong de Atari [WEB01]

El término *tiempo real* fue aplicado por primera vez a los gráficos por computadora que eran animados. Un uso más amplio de la palabra significa que la computadora puede procesar una entrada tan rápido o más de lo que la entrada es suministrada. Por ejemplo, hablar por teléfono es una actividad en tiempo real en la cual 2 personas participan. En la realidad, hay un retraso involucrado debido a la electrónica, pero el retraso es usualmente imperceptible para quienes mantienen la conversación. En contraste, el escribir una carta no es una actividad de comunicación en tiempo real.

[17] Pong fue publicado por Atari en 1972 y es considerado el primer videojuego
[18] Un tipo de mapa de bits dibujados en la pantalla de ordenador por hardware gráfico especializado

Aplicar el término tiempo real a los gráficos por computadora significa que la computadora esta produciendo una animación o secuencia de imágenes directamente en respuesta a alguna entrada, como el movimiento de un joystick[19], teclas presionadas y demás. Los gráficos por computadora en tiempo real pueden mostrar una forma de onda que esta siendo medida por equipos electrónicos, lecturas numéricas, o juegos interactivos y simulaciones visuales.

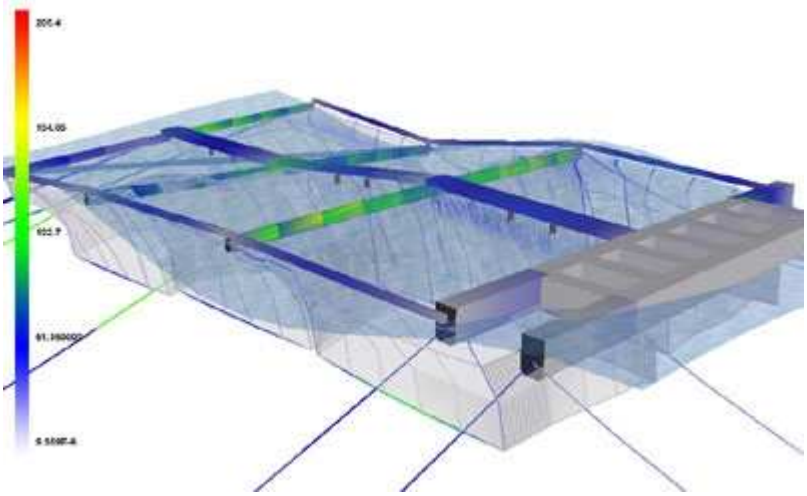


Ilustración 67: Simulación de Olas en granjas para peces [WEB13]

2.1.2 Gráficos en 3D

El término *tridimensional*, o 3D, significa que el objeto que esta siendo descrito o mostrado tiene tres dimensiones de medición: alto, largo y profundidad. Un ejemplo de un objeto bidimensional es una hoja de papel con un dibujo o un escrito en él, sin una profundidad perceptible. Un objeto tridimensional es una lata de jugo o gaseosa. La lata es redonda (ancho y profundidad) y alta. Dependiendo de su perspectiva, se puede alterar en cual lado de la lata esta el ancho o largo de esta, pero se mantiene el hecho de que tiene tres dimensiones.

[19] Dispositivo periférico para jugar con el ordenador

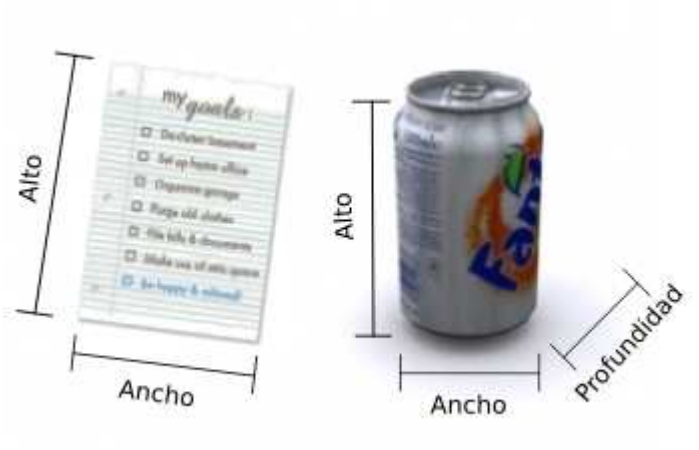


Ilustración 68: Diferencias entre un objeto de 2 dimensiones y un objeto tridimensional.

Por siglos, los artistas han sabido como hacer que una pintura parezca tener profundidad. Una pintura es inherentemente un objeto de bidimensional porque no es nada más que un lienzo con pintura en él. Similarmente, los gráficos 3D en computadora son actualmente imágenes bidimensionales en una pantalla plana que provén la ilusión de profundidad o la tercera dimensión.

2.1.3 2D + Perspectiva = 3D

Los primeros gráficos por computadora si duda fueron como en la Ilustración 69, donde se puede apreciar un cubo tridimensional simple dibujado con 12 segmentos de línea. Lo que hace ver al cubo tridimensional es la perspectiva, o el ángulo entre las líneas que crean la ilusión de profundidad.

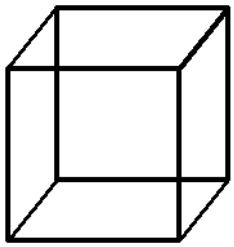


Ilustración 69: Un cubo simple.

Para realmente ver en 3D, se requiere ver un objeto con ambos ojos o suministrar a cada ojo imágenes separadas y únicas del objeto. Por ejemplo en la Ilustración 70 cada ojo recibe una imagen tridimensional que es como una

fotografía temporal mostrada en cada retina[20]. Estas dos imágenes son ligeramente diferentes porque estas son percibidas desde dos ángulos diferentes (nuestros ojos están separados a propósito). El cerebro luego, combina estas imágenes ligeramente diferentes para producir una única, imagen compuesta en 3D.

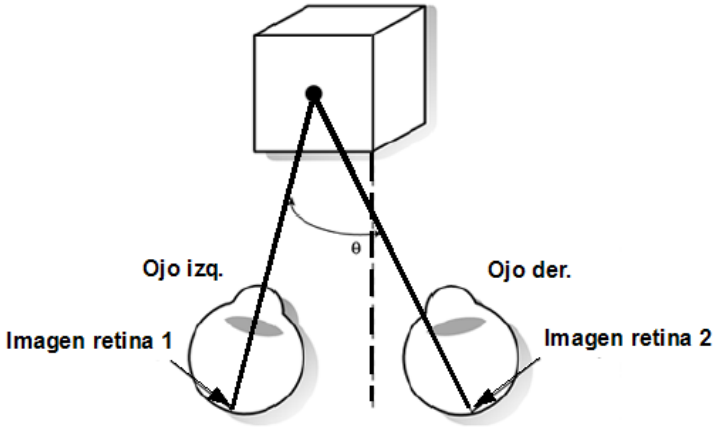


Ilustración 70: Como se ve en tres dimensiones [WRIG04].

En la Ilustración 70, el ángulo entre las imágenes se hace más pequeño mientras más se aleje el objeto del observador. Se puede amplificar este efecto 3D al incrementar el ángulo entre las dos imágenes. Las películas en 3D[21] explotan este efecto al colocar cada ojo en un lente separado o al proyectar dos imágenes súperimpuestas vistas a través de gafas o lentes de colores distintos.

Ilustración 71: Tipos de gafas para ver películas en 3D [WEB16].

[20] La retina es la capa más interna de las tres capas del globo ocular y es el tejido fotorreceptor.
[21] El 10 de junio de 1915 se produjo la primera proyección de una película en 3D ante público particular; el lugar fue el Teatro Astor, de New York.

Una pantalla de computador es una imagen plana en una superficie plana, no dos imágenes desde diferentes perspectivas que se aplican a cada ojo. Como resultado, mucho de lo que se considera ser gráficos computarizados en 3D es en realidad una aproximación de verdaderas tres dimensiones. Esta aproximación se logra en la misma manera en la que los artistas han creado sus pinturas con profundidad aparente por años, usando los mismos trucos que la naturaleza provee a las personas que tienen un solo ojo.

Es de notar que al cubrir uno de nuestros ojos, el mundo alrededor no se vuelve plano de repente. ¿Que pasa cuando nos cubrimos un solo ojo? Se puede pensar que todavía se ve en 3D, pero si colocamos un objeto dentro del alcance de nuestros brazos y luego cubrimos uno de nuestros ojos, al querer alcanzarlo existirán dificultades ya que la percepción de las distancias es afectada. Es por esto que las personas que tienen solo un ojo a menudo tienen dificultad en percibir distancias y profundidades.

La perspectiva es suficiente para crear la apariencia de tres dimensiones. Como en la Ilustración 72, inclusive sin color o sombreado (shading), el cubo aún tiene la apariencia de un objeto tridimensional. Si miramos al cubo por un tiempo prolongado, entonces el frente y el fondo del cubo parecen cambiar de lugar. Nuestro cerebro se confunde por la falta de información sobre las superficies de la figura.

En la Ilustración 72 se puede apreciar que el cubo descansando sobre un plano tiene una perspectiva exagerada pero aun puede producir el efecto de “saltar” hacia fuera de sus dos dimensiones.

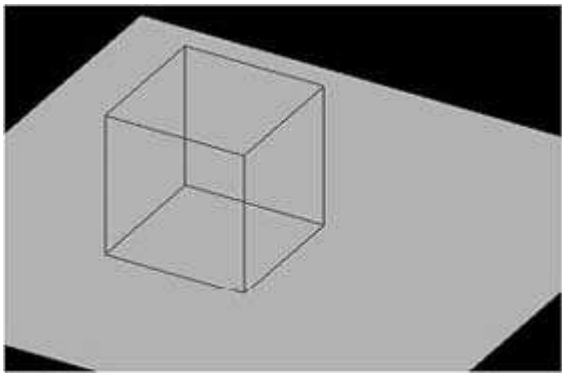


Ilustración 72: Un cubo 3D a base de líneas [WRIG04].

2.2 Artificios 3D

2.2.1 Introduccion

La razón por la cual el mundo no se vuelve plano de repente al cubrir uno de nuestros ojos es porque muchos de los efectos de un mundo 3D aun están presentes al ser vistos de manera bidimensional. Estos efectos son suficientes para disparar la habilidad del cerebro para percibir la profundidad. La pista más obvia es que los objetos cercanos parecen más grandes que los objetos más distantes. Este efecto de perspectiva es llamado foreshortening[22]. Este efecto cambia las texturas, iluminación, sombreado y las variaciones del color (debido a la iluminación) en conjunto y se suma a nuestra percepción de una imagen tridimensional.

Ahora es claro que la ilusión de 3D es creada en una pantalla de computadora al utilizar un conjunto de trucos artísticos y de perspectiva. A continuación se listarán estos efectos como una referencia a los que se revisará en las próximas secciones.

El primer término que se debe tomar en consideración es el renderizado (rendering). Rederizado es el acto de tomar una descripción geométrica de un objeto tridimensional y convertirlo en una imagen en pantalla de dicho objeto. Todos los siguientes efectos son aplicados al renderizar los objetos o una escena.

2.2.2 Perspectiva

La perspectiva se refiere a los ángulos entre las líneas que crean la ilusión de tres dimensiones. La Ilustración 73 muestra un cubo tridimensional dibujado sólo con líneas. Esta es una ilusión muy poderosa, pero aún puede causar problemas de percepción como se mencionó previamente. En la Ilustración 73, el cerebro tiene mas pistas de la verdadera orientación del cubo debido a la eliminación de líneas escondidas. Es de esperar que la parte frontal de un objeto obscurezca o cubra la parte trasera y esta no pueda ser vista. En

[22] Alteración de la escala de una imagen para sugerir una perspectiva.

superficies sólidas a esto se le llama *eliminación de superficies ocultas* (hidden surface removal).

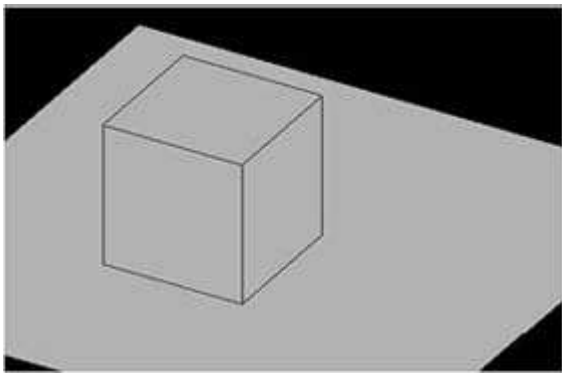


Ilustración 73: Un cubo 3D sólido más convincente [WRIG04].

2.2.3 Color y Sombreado

Si observamos al cubo de la Ilustración 73 por un tiempo prolongado, nos podemos convencer de que estamos mirando una imagen de un plano con un cubo enterrado o cavado y no a un cubo sobre una superficie. Para incrementar nuestra percepción, debemos ir mas allá del dibujado de solo líneas y agregar color para crear objetos sólidos.

La Ilustración 74 muestra qué sucede cuando nativamente agregamos color al cubo. Este ya no luce como un cubo. Al aplicar colores diferentes a cada lado, como en la Ilustración 75, volvemos a ganar percepción de un objeto sólido.

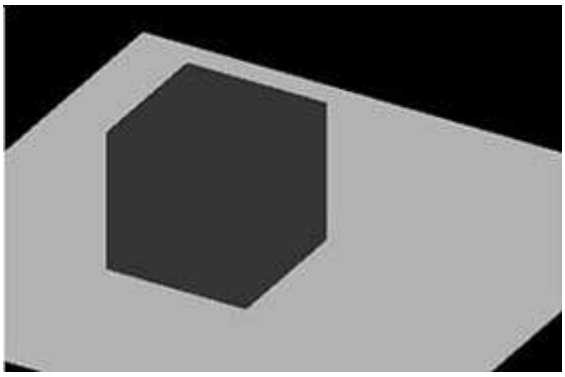


Ilustración 74: Un cubo 3D de un solo color [WRIG04].

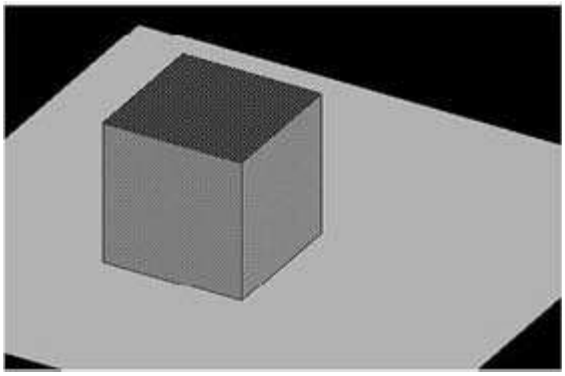


Ilustración 75: Un cubo 3D cuyas caras tienen distintos colores [WRIG04].

2.2.4 Luz y Sombras

Al hacer que cada lado del cubo tenga un color distinto ayuda a nuestros ojos a distinguir los distintos lados del objeto. Al sombrearlo apropiadamente podemos darle al cubo la apariencia de ser de un solo color (o material) pero también mostrar que esta siendo iluminado por una fuente de luz en un ángulo determinado, como se ve en la parte superior de la Ilustración 76. En la Ilustración 76 (inferior) se da un paso más allá al agregar una sombra tras el cubo. Ahora estamos simulando los efectos de la luz en uno o más objetos y sus interacciones. La ilusión creada hasta este punto es muy convincente.

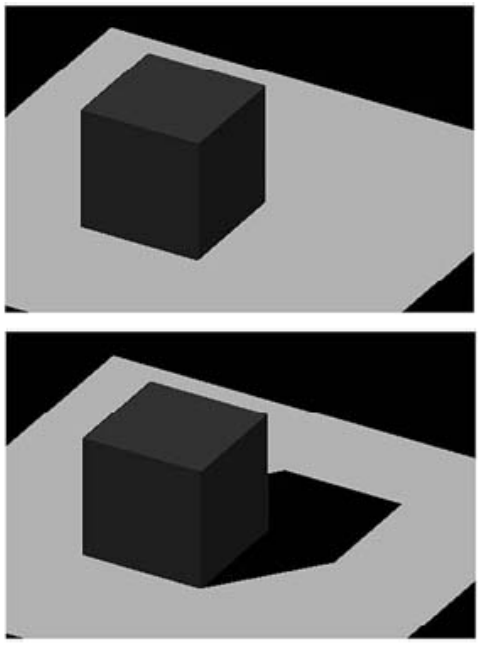


Ilustración 76: Agregando sombras para incrementar el realismo [WRIG04].

2.2.5 Mapeado de Texturas

Alcanzar un alto nivel de realismo sin nada más que miles o millones de pequeños polígonos iluminados y sombreados es materia de fuerza bruta y mucho trabajo duro. Desafortunadamente, mientras más geometría se envía al hardware grafico, más tiempo tomará en renderizarse. Una técnica inteligente permite usar geometría más simple pero alcanzar un alto nivel de realismo. Esta técnica toma una imagen, como una fotografía de una superficie real en detalle, y luego la aplica a la superficie de un polígono.

En vez de materiales con colores planos, se puede tener madera, ropa, ladrillos, baldosa, mármol, etc. Esta técnica de aplicar una imagen a un polígono para proveer detalle adicional de llama *mapeado de texturas* (texture mapping). La imagen provista es llamada *textura* (texture), y los elementos individuales de una textura son llamados texeles (texels). Finalmente, el proceso de comprimir o estirar los texeles sobre la superficie de un objeto de llama *filtrado (filtering)*. La Ilustración 77 muestra texturas aplicadas a los polígonos u objetos anteriormente mostrados.

Ilustración 77: La aplicación de texturas agrega más detalle sin geometrías adicionales.

2.2.6 Niebla

La mayoría de nosotros conoce la niebla o neblina. La niebla (fog) es un efecto atmosférico que agrega una falta de claridad a los objetos de una escena, la cual es usualmente la relación de cuan lejos los objetos en la escena están del

observador y cuan espesa es la niebla. Los objetos muy lejanos (o cercanos si la niebla es espesa) podrían ser totalmente obscurecidos.

Ilustración 78: El efecto de niebla provee la ilusión de espacios abiertos amplios [WRIG04].

2.2.7 Blending y Transparencia

Blending (mezclado) es una combinación de colores u objetos en una escena. Este efecto puede ser utilizado en una variedad de propósitos. Al variar la cantidad en la cual cada objeto es mezclado con la escena, se puede hacer que los objetos parezcan transparentes de manera que se puede apreciar el objeto y lo que esta detrás de él (como un vaso de vidrio o una imagen fantasmal).

También se puede usar blending para crear la ilusión de reflexión, como se muestra en la Ilustración 79. Se aprecia un cubo texturizado renderizado dos veces. Primero, el cubo es renderizado bajo el nivel del piso. Luego es piso es mezclado con la escena, permitiendo que es cubo sea visto a través de el. Finalmente, el cubo es dibujado sobre el nivel del piso. El resultado es la apariencia de reflexión en una superficie brillante de mármol.

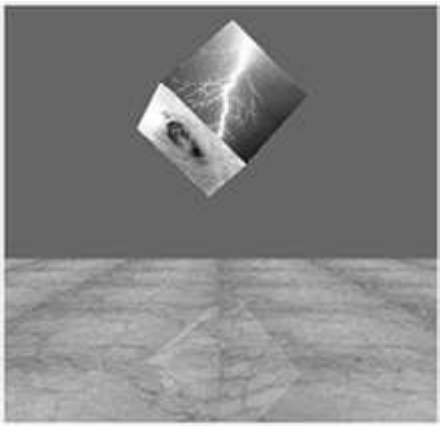


Ilustración 79: Ilusión de reflexión utilizando blending [WRIG04].

2.2.8 Antialiasing

Aliasing es un efecto visible en la pantalla debido al hecho de que una imagen consiste en píxeles discretos. En la Ilustración 80, se aprecia que las líneas tienen bordes cortantes. Al mezclar cuidadosamente las líneas con el color de fondo, se puede eliminar los bordes toscos y dar a las líneas una apariencia suavizada, como en la parte derecha de la Ilustración 80. Esta técnica es llamada suavizado o alisado (antialiasing). También es posible aplicar el suavizado a los bordes de polígonos para producir una escena más realista. Este “Santo Grial” de los gráficos en tiempo real también es llamado renderizado fotorealístico.

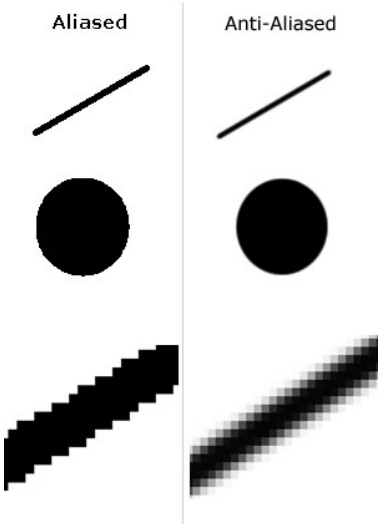


Ilustración 80: Aliasing vs Antialiasing

2.3 Usos comunes para los Gráficos 3D

Los gráficos tridimensionales tienen una gran variedad de usos en las aplicaciones computacionales modernas. Las aplicaciones de gráficos 3d en tiempo real varían desde juegos interactivos y simulaciones hasta la visualización de datos para objetivos científicos, médicos o de negocios. Los gráficos 3D de más alto nivel encuentran su camino en películas y publicaciones técnicas y educacionales.

2.3.1 3D en Tiempo Real

Como se definió anteriormente, los gráficos 3D en tiempo real son animados e interactivos con el usuario. Uno de los usos más recientes de los gráficos 3D en tiempo real son los simuladores de vuelo militares. Inclusive hoy, los simuladores de vuelo son populares para el usuario común como entretenimiento. La Ilustración 81 muestra una captura de pantalla de un simulador de vuelo que usa OpenGL para el renderizado 3D (www.flightgear.org).

Ilustración 81: Flight Gear, un popular simulador de vuelo basado en OpenGL.

Las aplicaciones para gráficos 3D en la computadora son casi ilimitadas. Talvez el uso más común el día de hoy es en los juegos de computadora o videojuegos. Difícilmente llega un título que no requiera de una tarjeta grafica

3D instalada en su PC para jugarlo. Los gráficos en 3D siempre han sido populares en las aplicaciones de visualización científica y de ingeniería, pero la explosión del hardware 3D barato ha motivado estas aplicaciones como nunca antes. Las aplicaciones de negocios también están tomando ventaja de la disponibilidad del hardware para incorporar más y más complejos gráficos de negocios y visualización de técnicas de minería de datos (datamining). Inclusive los modernos entornos gráficos están utilizando el hardware 3D como los escritorios 3D Beryl y Compiz-Fusion para Linux o el escritorio Aero de Microsoft Windows Vista, inclusive en Macintosh OS X, que también utiliza OpenGL para renderizar y gestionar las ventanas, todo esto con el propósito de brindar al usuario una nueva experiencia visual y poderosa.

Ilustración 82: El escritorio 3D Beryl para Linux.

Ilustración 83: El escritorio 3D Aero de Microsoft Windows Vista

Ilustración 84: El entorno Mac OSX.

2.3.2 3D en Tiempo no Real

Algún compromiso es requerido para las aplicaciones 3D de tiempo real. Al tener más tiempo de procesamiento, se puede generar gráficos 3D de altísima calidad. Típicamente, se diseña modelos y escenas, y un trazador de rayos (ray tracer) procesa la definición para producir una imagen 3D de alta calidad. El proceso típico es que alguna aplicación de modelado use gráficos 3D en tiempo real para interactuar con el artista para crear el contenido. Luego los cuadros o frames son enviados a otra aplicación (el trazador de rayos), la cual renderiza la imagen.

Por ejemplo, renderizar un solo frame o cuadro de una película animada como Toy Story podría tomar horas en una computadora muy rápida. El proceso de renderizar y guardar muchos miles de frames o cuadros genera una secuencia animada para una posterior reproducción. Aunque la reproducción pueda parecer de tiempo real, el contenido no es interactivo, así que no es considerado de tiempo real, sino pre-renderizado.

Ilustración 85: Imagen prerenderizada de un personaje de Final Fantasy XIII[23].

[23] Videojuego desarrollado por Square Enix para la consola PlayStation 3 de Sony. El personaje principal del juego es una bella mujer llamada Lighting.

2.4 Principios Básicos de la Programación 3D

Hasta ahora se ha cubierto las bases de los gráficos 3D en tiempo real y se ha cubierto ciertos términos importantes y algunos ejemplos de aplicaciones que utilizan gráficos de tiempo real. A continuación se describen las bases para iniciar el desarrollo de aplicaciones que usen gráficos en tres dimensiones.

2.4.1 Modo Inmediato y Modo Retenido

Existen dos diferentes modos en los API de gráficos 3D de tiempo real. El primero es llamado *modo retenido* [OPEN07]. En modo retenido, se provee al API o toolkit una descripción de los objetos y de la escena. El paquete grafico luego crea estas imágenes en la pantalla. La única manipulación adicional que se hace es dar comandos para cambiar la posición y la orientación del usuario (también llamada la *cámara*) u otros objetos en la escena.

Este tipo de modo es típico de los trazadores de rayos y muchos generadores de imágenes. Programadamente, la estructura que se construye es un grafo de escena. El grafo de escena es una estructura de datos (usualmente un GDA, o grafo dirigido acíclico) que contiene todos los objetos en la escena y las relaciones entre ellos. Muchos toolkits o "motores gráficos" usan este modelo. El programador no necesita entender los puntos complejos del renderizado, solo que tiene un modelo o base de datos que será manipulada a través de la librería de gráficos, la cual se encarga del renderizado.

El segundo alcance o modelo de renderizado 3D es llamado *modo inmediato* [OPEN07]. La mayoría de APIs de modo retenido o grafos de escena usan modo inmediato internamente para realizar el renderizado. En modo inmediato, no se describe modelos y/o el entorno desde alto nivel. En vez de eso, se ejecutan comandos directamente al procesador grafico que tiene un efecto inmediato en su estado y el estado de todos los comandos subsecuentes.

Con el modo inmediato, los nuevos comandos no tienen efecto en comandos de renderizado que ya se hayan ejecutado. Esto brinda mucho control de bajo nivel. Por ejemplo, se puede renderizar una serie de polígonos no iluminados

texturizados para representar el cielo. Luego ejecutar un comando para apagar la texturización, seguido por un comando para encender la iluminación. Entonces, toda la geometría que se dibuje (probablemente dibujada en el suelo) es afectada por la luz pero no es texturizada como el cielo.

2.4.2 Sistemas de Coordenadas

Antes de poder especificar la posición y tamaño de un objeto, se necesita un punto de referencia desde donde medirlo y colocarlo. Cuando se dibuja líneas o puntos en una pantalla de computador, se especifican las posiciones en términos de fila y columna. Por ejemplo, una pantalla estándar VGA tiene 640 píxeles desde la parte izquierda hasta la derecha y 480 píxeles de arriba hacia abajo. Para especificar un punto en el medio, se define la posición en (320,240), eso es 320 píxeles desde la izquierda y 240 píxeles hacia abajo desde la parte superior de la pantalla.

En OpenGL, o en casi cualquier API 3D, cuando se crea una ventana para dibujar en ella, también se especifica el sistema de coordenadas que se desea usar y cómo traducir las coordenadas especificadas en píxeles físicos en la pantalla. Primero veamos como se aplica estas bases en gráficos bidimensionales para luego entender esos principios para gráficos tridimensionales.

2.4.2.1 Coordenadas Cartesianas en 2D

El sistema más común de coordenadas es el Cartesiano [PAN96]. Las Coordenadas Cartesianas son especificadas por una coordenada X y una coordenada Y. La coordenada X es una medida de posición en la dirección horizontal, y la medida en la posición vertical es la coordenada Y.

El origen de un sistema Cartesiano esta en X=0, Y=0. Las Coordenadas Cartesianas son escritas como pares dentro de paréntesis, con la coordenada X en primer lugar y la coordenada Y en segundo lugar, separadas por una coma. Por ejemplo es origen del sistema se escribiría como (0,0). La Ilustración

86 muestra un sistema cartesiano en dos dimensiones. Las líneas gruesas X e Y son los ejes y se pueden extender positiva o negativamente hacia el infinito.

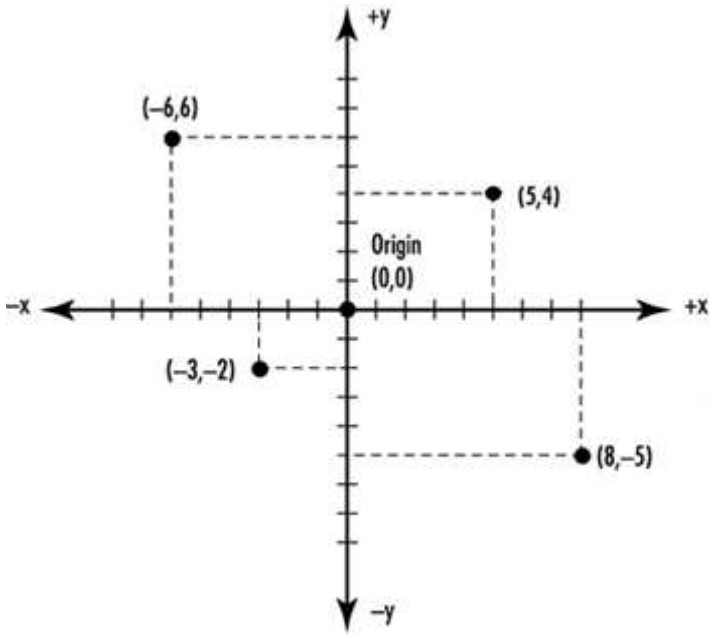


Ilustración 86: Sistema de Coordenadas Cartesianas [WRIG04].

El eje X y el eje Y son perpendiculares[24] y juntos definen el plano XY. Un plano es, de la manera más simple, una superficie plana. En cualquier sistema coordenado, dos ejes (o dos líneas) que intersectan en ángulos rectos definen un plano. En un sistema con solo dos ejes, naturalmente solo existe un plano.

2.4.2.2 Recortado de Coordenadas

Una ventana[25] es físicamente medida en términos de píxeles. Antes de poder empezar a dibujar en una ventana, hay que decir a OpenGL como traducir pares de coordenadas en coordenadas de pantalla. Esto se hace al especificar la región de espacio Cartesiano que ocupa la ventana; esta región es conocida como la región *clipping*. En el espacio bidimensional, la region de clipping[26] (recorte) son los valores mínimos y máximos de los valores X e Y que están dentro de la ventana. Otra forma de ver esto es especificando la posición del

[24] Dícese de la línea o plano que forma ángulo recto con otro
[25] Area visual de forma rectangular, que contiene algún tipo de interfaz de usuario
[26] Metodo de optimizacion utilizado en los gráficos por computadora donde solo se dibuja los objetos que son visibles al usuario.

origen en relación a la ventana. La Ilustración 87 muestra dos regiones de clipping comunes.

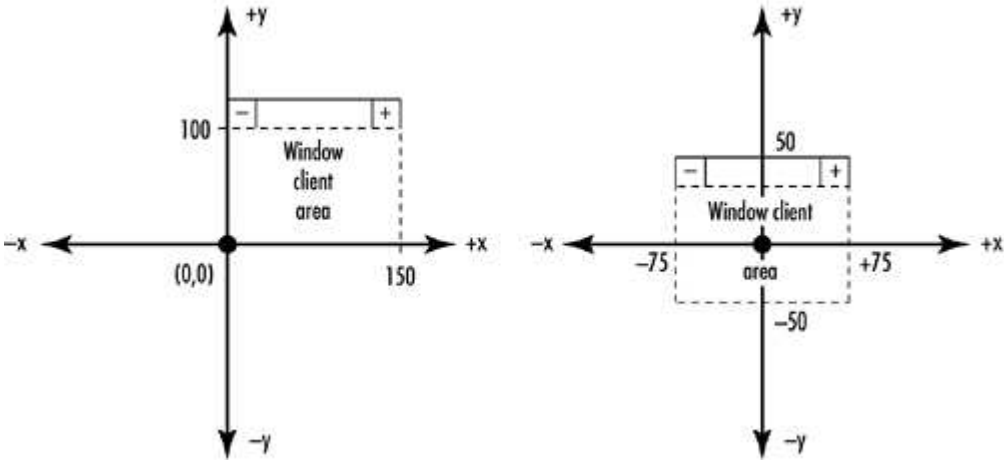


Ilustración 87: Regiones de Clipping comunes [WRIG04].

2.4.2.3 Conversión de Coordenadas de Dibujado a Coordenadas de Ventana

Rara vez el ancho y alto del área de clipping coincidirá con el alto y ancho de la ventana en píxeles. El sistema coordenado debe entonces ser transformado de coordenadas lógicas cartesianas a coordenadas físicas de píxeles en pantalla. Este mapeado es especificado por una configuración llamada *viewport* o puerto de vista. El viewport es la región dentro de la ventana del cliente que es usada para dibujar el area de clipping. El viewport simplemente mapea el area de recorte (clipping) hacia una región de la ventana. Usualmente, el viewport esta definido como la ventana completa, pero esto no es siempre estrictamente necesario; por ejemplo, se podría querer dibujar solamente en la mitad inferior de la ventana.

La Ilustración 88 muestra una ventana que mide 300x200 píxeles con un viewport definido como toda el área del cliente. Si el area de recorte para esta ventana tuviera desde 0 a 150 en el eje X y 0 a 100 en el eje Y, las coordenadas lógicas serian mapeadas en un sistema coordinado de pantalla mayor en la ventana. Cada incremento en el sistema coordenado lógico seria

empatado con dos incrementos en el sistema físico de coordenadas (píxeles) de la ventana.

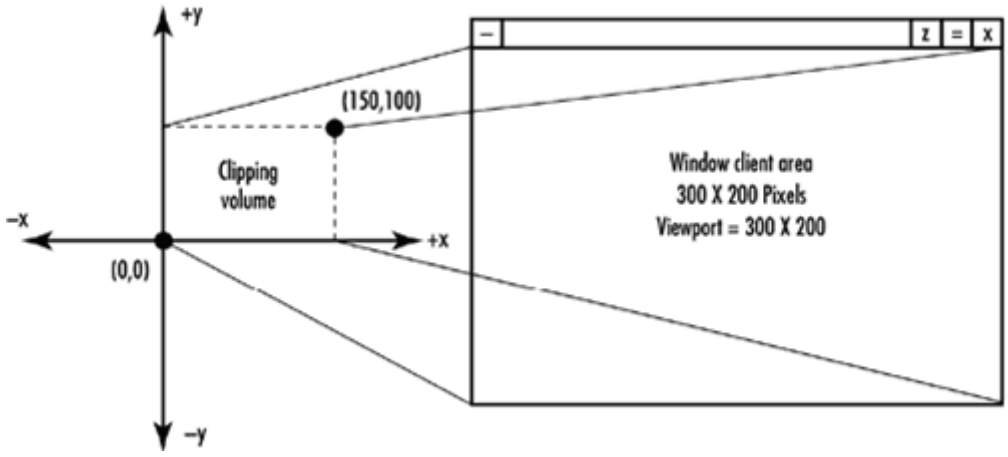


Ilustración 88: Un viewport definido como el doble del area de recorte [WRIG04].

En contraste, la Ilustración 89 muestra un viewport que coincide con el clipping área. La ventana aun es de 300x200 píxeles, sin embargo, esto causa que el área de visión ocupe solo la parte inferior izquierda de la ventana.

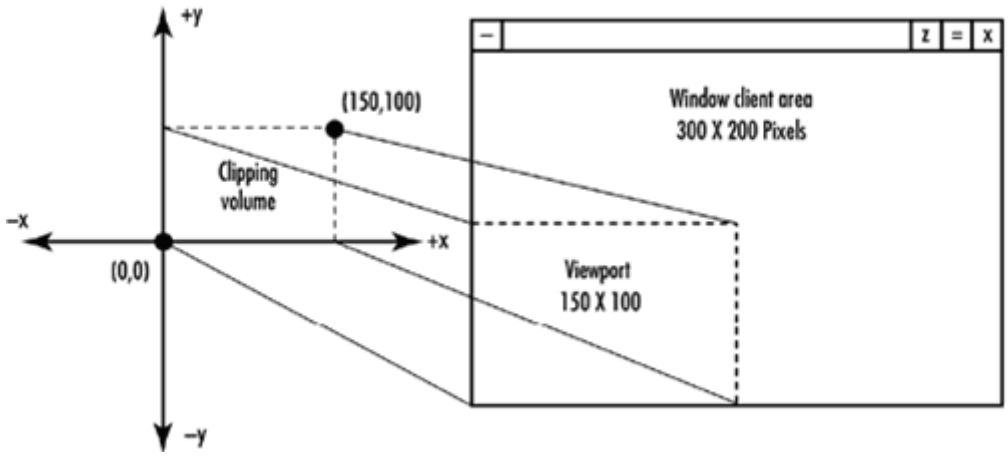


Ilustración 89: Un viewport que ocupa solo una parte de la ventana del cliente [WRIG04].

2.4.2.4 El Vértice, Una Posición en el Espacio

En 2D y 3D, cuando se dibuja un objeto, en realidad se lo compone de varias formas más pequeñas llamadas *primitivas*. Las primitivas son entidades bidimensionales o superficies como son los puntos, líneas y polígonos que son ensambladas en el espacio para crear objetos 3D. Por ejemplo, un cubo

tridimensional consiste de 6 cuadrados bidimensionales, cada uno colocado en una cada distinta. Cada esquina del cuadrado (o cualquier primitiva) es llamado *vértice* (vertex). Estos vértices luego son especificados para ocupar una coordenada particular en el espacio 3D. Un vértice no es nada más que una coordenada en un espacio 2D o 3D.

2.4.2.5 Coordenadas Cartesianas 3D

Ahora, extendemos nuestro conocido sistema coordenado hacia la tercera dimensión al agregar un componente de profundidad. La Ilustración 90 muestra el sistema Cartesiano con un nuevo eje, Z. El eje Z es perpendicular a los ejes X e Y. Representa una línea dibujada perpendicularmente desde el centro de la pantalla que se dirige hacia el observador. Ahora, especificamos una posición en tres dimensiones con tres coordenadas: X, Y, y Z. La Ilustración 90 muestra el punto (-4, 4,4) como ejemplo.

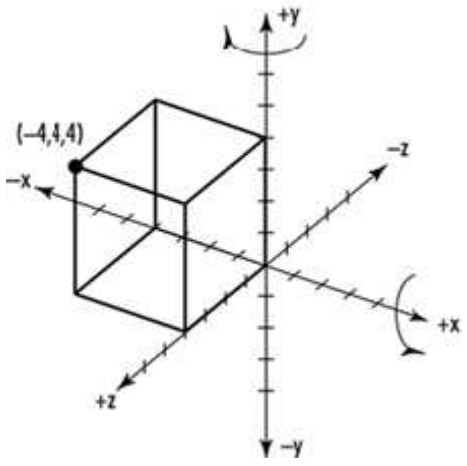


Ilustración 90: Sistema Coordenado Cartesiano de 3 dimensiones [WRIG04].

2.4.3 Proyecciones

Hemos visto cómo especificar una posición en espacio 3D usando coordenadas Cartesianas. No importa como queramos convencer al ojo humano, los píxeles en una pantalla solo tienen dos dimensiones. OpenGL traduce coordenadas Cartesianas en coordenadas bidimensionales a través de trigonometría y manipulación de matrices. No es algo simple; se podría realizar

una tesis completa sobre este tema. Afortunadamente no es necesario un conocimiento profundo de matemáticas para usar OpenGL y crear gráficos.

El primer concepto realmente necesario es la *proyección*. Las coordenadas 3D usadas para crear geometría son aplanadas o proyectadas en una superficie 2D (el fondo de la ventana). Es como trazar el contorno de un objeto tras un cristal con un marcador. En la Ilustración 91, una casa es trazada en un cristal. Al especificar la proyección, se especifica el volumen de visión que se desea desplegar en la ventana y cómo esta debería ser transformada.

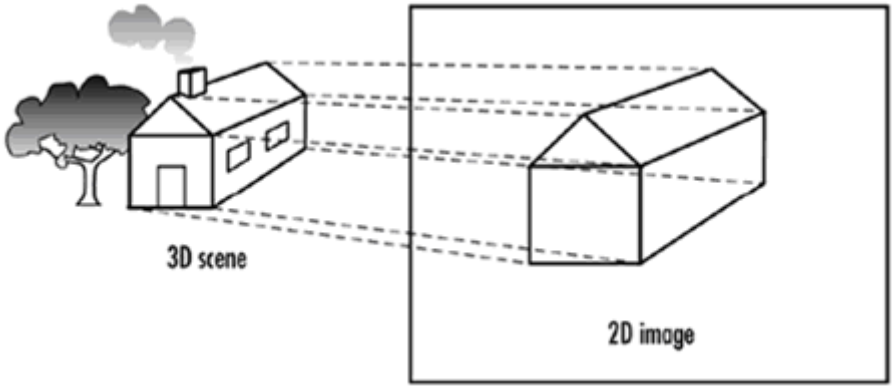


Ilustración 91: Una imagen 3D proyectada en una superficie 2D

2.4.3.1 Proyecciones Ortográficas

En OpenGL hay que tener en cuenta de dos tipos de proyecciones. La primera es llamada proyección ortográfica o paralela. Esta proyección es definida por un volumen de visión cuadrada o rectangular. Cualquier objeto fuera de este volumen no es dibujado. Además, todos los objetos que tienen las mismas dimensiones aparentan un mismo tamaño, independientemente si están muy cerca o muy lejos. Este tipo de proyección (mostrada en la Ilustración 92) es usada a menudo en diseño arquitectónico, diseño asistido por computadora (CAD), o gráficos 2D. Frecuentemente, la proyección ortográfica se utiliza para agregar texto en escenas de gráficos en 3D.

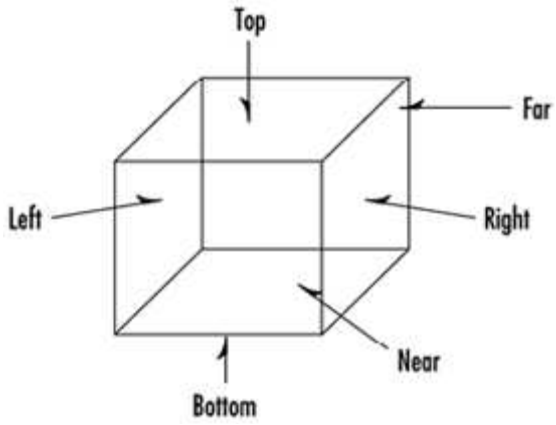


Ilustración 92: Volumen de la proyeccion ortografica

Se especifica el volumen de visión en una proyección ortográfica al definir los planos de recorte far (lejano), near (cercano), left (izquierda), right (derecha), top (cima), y bottom (fondo). Los objetos y figuras colocadas dentro de este volumen de visión luego son proyectadas (tomando en cuenta su orientación) en una imagen 2D que aparece en pantalla.

2.4.3.2 Proyecciones de Perspectiva

La segunda y más comúnmente usada proyección es la de perspectiva. Esta proyección agrega el efecto de que los objetos distantes, parezcan más pequeños que los más cercanos. El volumen de visión (véase la Ilustración 93) es parecido a una pirámide recortada su punta. La parte que queda es llamada frustum. Los objetos cercanos al frente del volumen de visión parecen mas cercanos a su tamaño original, pero los objetos cercanos al fondo del volumen se encojen al ser proyectados hacia el fondo del volumen. Este tipo de proyección da más realismo para la simulación y animación 3D.

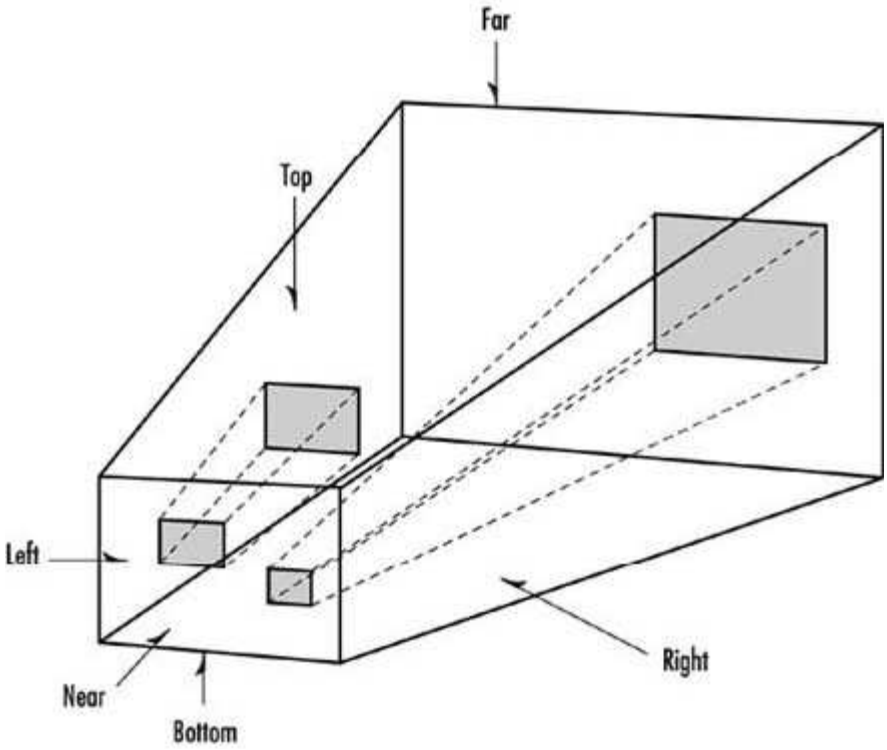


Ilustración 93: El volumen (frustum) de una perspectiva de proyección.

2.5 Introducción a OpenGL

2.5.1 ¿Qué es OpenGL?

OpenGL es definido estrictamente como "una interfaz software al hardware gráfico" [OPEN07]. En esencia, es una librería de gráficos y modelado 3D que es altamente portable[27] y muy rápida. Usando OpenGL, se puede crear gráficos 3D elegantes y sofisticados con la calidad visual de un trazador de rayos. Inicialmente usaba algoritmos desarrollados cuidadosamente y optimizados por Silicon Graphics[28], Inc. (SGI), un reconocido líder mundial en gráficos por computadora. OpenGL ha evolucionado gracias a contribuciones de otros vendedores que han brindado su experiencia y propiedad intelectual para desarrollar implementaciones de más alto desempeño.

OpenGL proviene del inglés Open Graphics Library [OPEN07], “Open” debido a que es un estándar abierto, de manera que las compañías pueden contribuir con su desarrollo, esto no significa que OpenGL sea Open Source[29].

OpenGL no es un lenguaje de programación como C o C++. Es más bien como la librería de ejecución de C, la cual provee funcionalidad precompilada. No existe algo como un "programa OpenGL," sino un programa que un desarrollador escribió que usa OpenGL como una de las Interfaces de Programación de Aplicaciones (APIs). Se podría usar el API de Windows para acceder a un archivo o la Internet, y podría usarse OpenGL para crear gráficos 3D en tiempo real [WRIG04].

OpenGL es intencionado para el uso con el hardware que es diseñado y optimizado para la manipulación de gráficos 3D. Implementaciones solo por software o genéricas de OpenGL también están disponibles, y las implementaciones de Microsoft caen en esta categoría [WRIG04]. Con una implementación solo por software, el renderizado no se realiza tan rápido, y

[27] Un programa informático (aplicación, o sistema operativo) es portable cuando se funciona en otros sistemas distintos a aquel en el que fue diseñado.
[28] Compañía fundada por Jim Clark y Abbey Silverstone en 1982.
[29] Código abierto (Open Source) es el término con el que se conoce al software distribuido y desarrollado libremente.

algunos efectos especiales avanzados no están disponibles. Sin embargo, usar una implementación por software, significa que los programas pueden ejecutarse en una gran variedad de sistemas que pudieran no tener tarjetas graficas 3D instaladas.

OpenGL al ser una interfaz independiente del hardware pude ser implementada en muchas diferentes plataformas. Para alcanzar este objetivo, no hay comandos incluidos para realizar tareas de ventanas, controlar los eventos o la entrada del usuario; en vez de eso, es necesario manipular estas acciones a través del sistema de control de ventanas del sistema que se este usando para ese hardware en particular.

De manera similar, OpenGL no provee comandos de alto nivel para describir modelos de objetos tridimensionales. Dichos comandos permitirían especificar formas relativamente complejas como automóviles, partes del cuerpo humano, aeroplanos o moléculas. Con OpenGL, es necesario construir el modelo deseado desde un reducido grupo de primitivas geométricas (puntos, líneas y polígonos). Una librería sofisticada que provea estas características se podría construir en base a OpenGL. GLU (OpenGL Utility Library) provee muchas características de modelado, como superficies y curvas NURBS. GLU es una parte estándar de toda implementación de OpenGL.

OpenGL es utilizado en una variedad de propósitos, desde ingeniería CAD (Diseño Asistido por Computador) y aplicaciones arquitectónicas hasta programas de modelado usados para crear criaturas generadas por computadora para películas para el cine y la televisión[30]. La introducción de un API 3D estándar en la industria al los sistemas de mercado masivo como Microsoft Windows y Macintosh OS X ha traído repercusiones muy interesantes. Con la aceleración por hardware y los microprocesadores más rápidos cada día, los gráficos 3D son ahora componentes típicos de las aplicaciones de consumo y de negocios, no solo de videojuegos o aplicaciones científicas.

[30] Jurassic Park, Star Wars, Matriz, King Kong son algunos ejemplos de películas que utilizaron los graficos por computadora para varios efectos especiales.

2.5.2 Breve historia de OpenGL

Al principio de los años 1990 SGI (Silicon Graphics Inc.) era un grupo de referencia en gráficos 3D para estaciones de trabajo. Suya era la API IRIS GL, considerada puntera en el campo y estándar de facto, llegando a eclipsar a PHIGS[31], basada en estándares abiertos. IRIS GL se consideraba más fácil de usar y, lo más importante, soportaba renderizado en modo inmediato. Además, PHIGS, aparte de su mayor dificultad, fue considerada inferior a IRIS GL respecto a funcionalidad y capacidad.

La competencia de SGI (Sun Microsystems, Hewlett-Packard e IBM, entre otros) fue capaz de introducir en el mercado hardware 3D compatible con el estándar PHIGS mediante extensiones. Esto fue reduciendo la cuota de mercado de SGI conforme iban entrando diferentes proveedores en el mercado. Por todo ello, en un intento de fortalecer su influencia en el mercado, SGI decidió convertir el estándar IRIS GL en un estándar abierto. SGI observó que la API IRIS GL no podía ser abierta debido a conflictos de licencias y patentes; también contenía funciones no relevantes para los gráficos 3D como APIs para ventanas, teclado o ratón (en parte, porque fue desarrollada antes de la aparición del X Window System[32]). Además, mientras iba madurando el soporte del mercado para el nuevo estándar, se pretendía mantener los antiguos clientes mediante bibliotecas añadidas como Iris Inventor[33] o Iris Performer. El resultado de todo lo anterior fue el lanzamiento del estándar OpenGL.

[31] PHIGS (Programmer's Hierarchical Interactive Graphics System) es un API estándar para hacer gráficos por ordenador en 3D, considerado como el estándar de gráficos 3D en los años 90.
[32] El sistema de ventanas X fue desarrollado a mediados de los años 1980 en el MIT para dotar de una interfaz gráfica a los sistemas Unix.
[33] Es un API 3D escrito en C++ orientado a objetos y en modo retenido diseñado por SGI para proveer un nivel mas alto de programación para OpenGL.

Algunos de los logros que se consiguieron fueron:

- Estandarizar el acceso al hardware.

- Trasladar a los fabricantes la responsabilidad del desarrollo de las interfaces con el hardware.

- Delegar las funciones para ventanas al sistema operativo.

- Con la variedad de hardware gráfico existente, lograr que todos hablasen el mismo lenguaje obtuvo un efecto importante, ofreciendo a los desarrolladores de software una plataforma de alto nivel sobre la que trabajar.

En 1992, SGI lideró la creación del OpenGL Architecture Review Board (OpenGL ARB), un grupo de empresas que mantendría y extendería la especificación OpenGL en los años siguientes [WRIG04]. OpenGL evolucionó desde IRIS GL, superando su problema de dependencia del hardware al ofrecer emulación software para aquellas características no soportadas por el hardware del que se dispusiese. Así, las aplicaciones podían utilizar gráficos avanzados en sistemas relativamente poco potentes.

En 1995 Microsoft lanzó Direct3D, que se convertiría en el principal competidor de OpenGL. El 17 de diciembre de 1997 Microsoft y SGI iniciaron el proyecto Fahrenheit, esfuerzo cooperativo con el objetivo de unificar las interfaces de OpenGL y Direct3D (y añadir también una API scene-graph). En 1998 se uniría al proyecto Hewlett-Packard. Pese a tener un principio prometedor en estandarizar las APIs de gráficos 3D, debido a restricciones financieras en SGI y la falta general de apoyo por parte de la industria, fue finalmente abandonado en 1999. Microsoft, uno de los miembros fundadores, abandonó el proyecto en 2003.

El 31 de julio de 2006 se anunció que el control de OpenGL pasaría del ARB al Grupo Khronos[34]. Con ello se intentaba mejorar el marketing de OpenGL y eliminar las barreras entre el desarrollo de OpenGL y OpenGL ES[35]. ARB se convirtió dentro de Khronos en el OpenGL ARB Working Group. El gran número de empresas con variados intereses que han pasado tanto por el antiguo ARB como por el grupo actual han hecho de OpenGL una API de propósito general con un amplio rango de posibilidades.

En 2006, algunos de los miembros del OpenGL ARB Working Group [WEB19] eran:

- AMD
- Creative Labs
- Intel Graphics Controllers
- nVIDIA
- Sony Computer Entertainment Inc.
- Sun Microsystems
- Texas Instruments

2.5.3 Estructura Básica de una Aplicación OpenGL

Debido a que se puede hacer muchas cosas con OpenGL, una aplicación que use OpenGL puede llegar a ser muy complicada. Sin embargo, la estructura básica de un programa útil puede ser muy simple: Inicializar los estados de control y especificar los objetos que deben ser renderizados.

[34] Es un consorcio fundado por miembros de la industria multimedia enfocado a la creación de estandares abiertos. www.khronos.org/

[35] OpenGL ES (OpenGL for Embedded Systems) es un subconjunto del API de OpenGL diseñado para dispositivos embebidos como telefonos celulares, PDAs y consolas de videojuegos.

Ejemplo 2: Estructura básica de un programa con OpenGL

```
#include <LibreriasNecesarias.h> /* se incluyen las cabeceras */
main() {  /* Este ejemplo dibuja un cuadrado blanco */
  InicializarVentana(); /* inicializaciones del gestor de ventanas*/
  glClearColor (0.0, 0.0, 0.0, 0.0); /* el color de fondo es negro */
  glClear (GL_COLOR_BUFFER_BIT);   /* se limpia el buffer de color */
  glColor3f (1.0, 1.0, 1.0);  /* el color para dibujar es blanco */
  glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0); /* perspectiva ortogonal */
  glBegin(GL_POLYGON);   /* se inicia el dibujado de un poligono */
    glVertex3f (0.25, 0.25, 0.0); /* primer vértice */
    glVertex3f (0.75, 0.25, 0.0); /* segundo vértice */
    glVertex3f (0.75, 0.75, 0.0); /* tercer vértice */
    glVertex3f (0.25, 0.75, 0.0); /* cuarto vértice */
  glEnd();             /* se da por terminado el poligono */
  glFlush();           /* se forza la culminacion del renderizado */
  ActualizarVentanaYEscucharEventos(); /* se gestionan eventos */
}
```

2.5.4 Tipos de Datos

Para facilitar la portabilidad del código OpenGL, se ha definido tipos de datos propios. Estos tipos de datos son similares a los de C y pueden ser usados si se desea. Sin embargo los varios compiladores y entornos, tienen sus propias reglas para el tamaño y la memoria de los varios tipos de datos de C. Al usar los tipos de datos de OpenGL, se puede aislar el código de estos tipos de cambios.

La Tabla 17 muestra los tipos de datos de OpenGL, su equivalencia en C para Windows (Win32), y el sufijo usado. Estos sufijos son utilizados por conveniencia para muchas funciones OpenGL [OPEN07].

Tabla 17: Tipos de datos disponibles en OpenGL [OPEN07]

Sufijo	Tipo de Dato	Equivalencia en C	Definición OpenGL
b	Entero de 8 bits	char con signo	GLbyte
s	Entero de 16 bits	short	GLshort
i	Entero de 32 bits	int, long	GLint, GLsizei

f	Punto flotante de 32 bits	float	GLfloat, GLclampf
d	Punto flotante de 64 bits	double	GLdouble, GLclampd
ub	Entero sin signo de 8 bits	char sin signo	GLubyte, GLboolean
us	Entero sin signo de 16 bits	short sin signo	GLushort
ui	Entero sin signo de 32 bits	int sin signo, long sin signo	GLuint, GLenum, GLbitfield

Todos los tipos de datos empiezan con GL para denotar que es de OpenGL. La mayoría incluye el tipo correspondiente de C (byte, short, int, float). Algunos tienen la u al inicio para denotar unsigned (sin signo o positivo). También existen descripciones mas precisas como size par el tamaño del valor a almacenar. La designación clamp es una pista de que los valores serán ajustados (clamped) en el rango 0.0–1.0.

2.5.5 Conveniencias del Nombre de las Funciones

La mayoría de funciones de OpenGL siguen reglas de descripción de nombres que indican de qué librería es dicha función, cuántos y que tipos de datos toma como argumentos [OPENG07]. Todas las funciones tienen una raíz que indica el comando correspondiente de OpenGL. Por ejemplo, glColor3f tiene de raíz a *Color*. El prefijo *gl* indica la librería, y el sufijo *3f* significa que la función toma tres argumentos de punto flotante. Todas las funciones OpenGL siguen el siguiente formato:

```
<Prefijo de Librería><Comando raíz><Número de Argumentos
  (Opcionales)><Tipo de Dato del Argumento (opcional)>
```

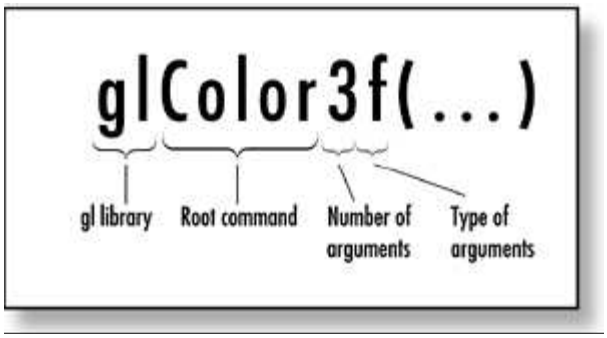


Ilustración 94: Formato de las funciones de OpenGL [WRIG04].

2.5.6 Librerías y Cabeceras

Aunque OpenGL es un "estándar", esta librería tiene muchas implementaciones. Microsoft Windows soporta OpenGL mediante software (opengl32.dll). En la mayoría de plataformas, la librería OpenGL es acompañada por GLU (glu32.dll en Windows).

Los pasos para configurar un compilador para vincular a OpenGL varían de herramienta en herramienta y de plataforma a plataforma.

Por ejemplo, la forma típica de las inclusiones de un programa para Windows que usa OpenGL se muestra a continuación:

```
#include<windows.h>
#include<gl/gl.h>
#include<gl/glu.h>
```

Si se utiliza GLUT[36] para el manejo de las tareas de gestión de ventanas se debería incluir:

```
#include <GL/glut.h>
```

Nótese que glut.h incluye gl.h, glu.h, y glx.h automáticamente, entonces el incluir todos los tres archivos es redundante [OPEN07].

[36] GLUT (OpenGL Utility Toolkit) es una librería de utilidades para programas OpenGL que principalmente proporciona diversas funciones de entrada/salida con el sistema operativo.

2.5.7 El Renderizado en OpenGL

La mayoría de las implementaciones de OpenGL tienen un orden similar de operaciones, una serie de etapas de procesos llamados *pipeline*. Este orden, mostrado en la Ilustración 95, no es una regla estricta de cómo OpenGL es implementado, sino provee una guía confiable para predecir que es lo que OpenGL realizará.

El siguiente diagrama muestra la línea de ensamblaje de Henry Ford[37], que OpenGL usa para procesar los datos. Los datos geométricos (vértices, líneas, y polígonos) siguen el camino a través de cada proceso que incluye evaluadores y operaciones por-vértice, mientras que los datos de píxeles (píxeles, imágenes, y mapas de bits) son tratados diferente como parte del proceso. Ambos tipos de datos experimentan los mismos pasos finales (rasterización y operaciones por-fragmento) antes de que el ultimo píxel sea escrito en el *framebuffer*[38].

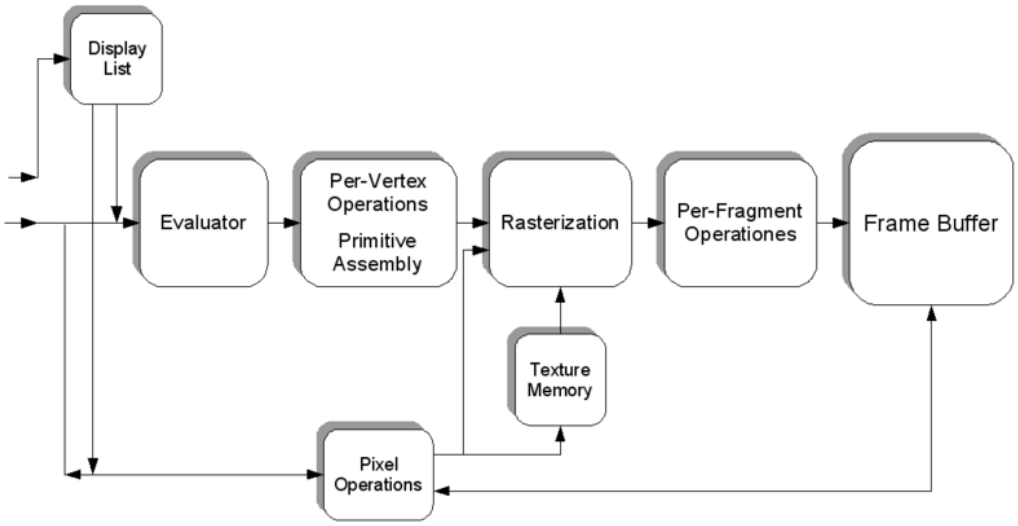


Ilustración 95: Orden de las operaciones de OpenGL [WEB01]

A continuación se detalla los procesos del pipeline de renderizado de OpenGL.

[37] Henry Ford (Dearborn, Michigan, 30 de julio de 1863 - 7 de abril de 1947) fue un industrial estadounidense, fundador de la compañía Ford Motor Company y padre de las cadenas de producción modernas utilizadas para la producción en masa.
[38] Es un dispositivo de video que manipula lo que se muestra desde un buffer de memoria que contiene los datos del frame o cuadro completo.

2.5.7.1 Lista de Visualización

Todos los datos (geométricos o píxeles), pueden ser almacenados en una lista para su utilización. (La alternativa a almacenar los datos en una lista seria procesar los datos inmediatamente en el modo inmediato). Cuando una lista de visualización es ejecutada, los datos retenidos son enviados como si fueran enviados por la aplicación en modo inmediato [OPEN07].

2.5.7.2 Evaluadores

Todas las primitivas geométricas son eventualmente descritas por vértices. Las curvas paramétricas y superficies pueden ser inicialmente descritas por puntos de control y funciones polinomiales llamadas funciones básicas. Los evaluadores proveen un método para derivar los vértices usados para representar la superficie desde los puntos de control. El método es mapeado polinomial, el cual puede producir las normales de una superficie, coordenadas de textura, colores y valores de coordenadas espaciales desde los puntos de control [OPEN07].

2.5.7.3 Operaciones Por-Vértice

Para los vértices, sigue la etapa de "operaciones por-vértice", la cual convierte los vértices en primitivas. Algunos vértices (por ejemplo coordenadas espaciales) son trasformadas a matrices de punto flotante de 4x4. Estas coordenadas espaciales son proyectadas desde una posición en el mundo 3D hacia la pantalla.

Si características mas avanzadas son activadas, esta etapa se vuelve más compleja. Si se activa texturización, las coordenadas de textura pueden ser generadas y transformadas aquí. Si la iluminación es activada, los cálculos de iluminación son realizados usando el vértice ya transformado, la normal de la superficie, la posición de la fuente de luz, las propiedades de los materiales, y otra información de iluminación para producir un valor de color [OPEN07].

2.5.7.4 Ensamblado de Primitivas

La mayor parte del ensamblado de primitivas, es la eliminación de porciones de geometría que salen de un espacio definido por un plano. El clipping de puntos pasa o rechaza vértices; clipping de líneas o polígonos puede agregar vértices adicionales dependiendo de cómo la línea o el polígono es recortado [OPEN07].

En algunos casos, esto es consecutivo por división de perspectiva, la cual hace que los objetos distantes aparezcan mas pequeños que los objetos mas cercanos. Luego las operaciones del viewport y profundidad (coordenada Z) son aplicadas. Si culling[39] esta activado y la primitiva es un polígono, este puede ser rechazado por una prueba de culling. Dependiendo del modo del polígono, un polígono pude ser dibujado como líneas o puntos.

Los resultados de esta etapa son primitivas geométricas completas, las cuales resultan de los vértices transformados y recortados con un color, profundidad y algunas veces valores de coordenadas de textura similares y guías para el siguiente paso de rasterización [OPEN07].

2.5.7.5 Operaciones de Píxeles

Mientras los datos geométricos toman un camino a través del pipeline, los datos de píxeles toman una ruta distinta. Los píxeles de un array en la memoria del sistema primero son desempacados de una de las variedades de formatos en un número apropiado de componentes. Luego los datos son escalados y procesados por un mapa de píxeles. Los resultados luego son ajustados (clamped) y escritos en la memoria de texturas o enviados al paso de rasterización.

Si los datos de los píxeles son leídos desde el framebuffer, las operaciones de trasferencia de píxeles son ejecutadas (escalar, mapear y ajustar). Luego los

[39] Eliminación del pipeline de los objetos que no pueden ser vistos desde el punto de vista.

resultados son empacados en un formato apropiado y retornados a un array en memoria del sistema [OPEN07].

2.5.7.6 Ensamblado de Texturas

Una aplicación OpenGL podría requerir la aplicación de texturas en objetos geométricos para que estos parezcan más reales. Si varias imágenes de textura son usadas, es útil ponerlas en objetos de textura para poder cambiar de una a otra fácilmente [OPEN07].

2.5.7.7 Rasterización

Rasterización es la conversión de datos geométricos y píxeles en fragmentos. Cada fragmento corresponde a un píxel en el framebuffer. Tipos de líneas y polígonos, ancho de línea, tamaño de un punto, sombreado de modelos y cálculos de suavizado son tomados en consideración mientras los vértices son conectados como líneas o los píxeles interiores son calculados para rellenar un polígono. Los valores de color y profundidad son asignados para cada fragmento [OPEN07].

2.5.7.8 Operaciones de Fragmentos

Antes de que los valores sean guardados en el framebuffer, una serie de operaciones son realizadas que pueden alterar o descartar fragmentos. Todas estas operaciones pueden ser activadas o desactivadas. La primera operación es texturizado, donde un texel (un elemento de textura) es generado desde la memoria de texturas para cada fragmento y aplicado al fragmento. Luego los cálculos de niebla son aplicados, seguidor por la prueba de tijeras, la prueba alfa, la prueba de stencil[40], y la prueba de profundidad de buffer (el buffer de profundidad es para eliminación de superficies ocultas).

[40] El stencil es una máscara que se aplica a la pantalla para delimitar zonas que se renderizarán y otras que no cambiaran de estado. Esto permite renderizar sólo una porción de la pantalla.

Si una prueba falla termina el proceso del fragmento. Luego, blending (mezclado), dithering[41], operaciones lógicas, y enmascarado por un mapa de bits pueden ser realizados. Finalmente, el fragmento procesado es dibujado en el buffer apropiado, donde ya es un píxel y ha alcanzado su lugar final de descanso [OPEN07].

[41] El proceso según el cual se consigue adaptar un color a los píxeles adyacentes para simular un tercer color en una imagen de mapa de bits

2.6 OpenGL como una Máquina de Estados

Aunque se puede dibujar imágenes complejas e interesantes usando OpenGL, todas son construidas desde un determinado número de primitivas graficas. Esto no debería de sorprender, grandes artistas crearon obras maestras con solo pinceles y oleos.

En el nivel máximo de abstracción, hay tres operaciones básicas para dibujar: limpiar o borrar la ventana, dibujar un objeto geométrico, y dibujar un objeto ráster. Los objetos ráster incluyen imágenes bidimensionales, fuentes de caracteres y mapas de bits.

Como se explico anteriormente, los gráficos por computadora, como por ejemplo los que se ven en el cine o la televisión no son nada mas que millones de polígonos o líneas que en conjunto crean superficies muy complejas.

2.6.1 Generalidades del Dibujo en Computador

Al dibujar en una pantalla de computadora, la memoria que contiene las imágenes es usualmente llenada con la última imagen que se dibujo, así que típicamente se necesita vaciarla o borrarla con un determinado color de fondo antes de dibujar la siguiente imagen. El color de fondo depende de la necesidad que se tenga, por ejemplo, un fondo negro para el espacio exterior o un fondo azul cielo para un escenario de un día despejado.

Pero, ¿porque no solamente dibujar un rectángulo que cubra toda la pantalla para borrarla? Pues, un comando especial para limpiar la pantalla es mucho más eficiente que un comando de propósito general para la misma tarea [OPEN07].

También hay que conocer que los colores de los píxeles se guardan en la memoria del hardware grafico llamado bitplanes. Existen dos modos de almacenamiento. Los valores RGBA (rojo, verde, azul, alfa) de un píxel pude ser almacenado directamente en los bitplanes, o un valor índice que hace

referencia a ese color en una tabla. El modo RGBA es el más comúnmente usado, aunque también en ocasiones es necesario el modo de color indexado.

Por ejemplo, para limpiar una ventana en modo RGBA se usan estas líneas de código OpenGL:

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClear(GL_COLOR_BUFFER_BIT);
```

La primera línea define el color negro como color de fondo o de borrado, y el siguiente comando, limpiará toda la pantalla con el color actual.

El único parámetro de glClear() indica los buffers que se van a limpiar. Típicamente se indica el color de fondo una sola vez al inicio del programa y se limpia los buffers cuando se necesite. OpenGL mantiene el color actual como una variable de estado hasta que esta sea cambiada [OPEN07].

Ahora, si se desea limpiar otros buffers, se usaría los comandos ya conocidos:

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClearDepth(1.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Ahora el comando glClear() recibe parámetros unidos por el OR lógico para limpiar todos los buffers definidos.

Tabla 18: Lista de Buffers que se pueden limpiar con glClear() [OPEN07].

Buffer	Nombre
Buffer de color	GL_COLOR_BUFFER_BIT
Buffer de Profundidad	GL_DEPTH_BUFFER_BIT
Buffer de Acumulación	GL_ACCUM_BUFFER_BIT
Buffer Stencil	GL_STENCIL_BUFFER_BIT

2.6.1.1 Especificando un Color

En OpenGL, la descripción de un objeto es independiente de su color. Cuando se dibuja un objeto geométrico, este es dibujado usando el esquema de color actual. Este esquema puede ser tan sencillo como dibujar todos los objetos en rojo vivo, o puede ser tan complicado como asumir que el objeto es de plástico y hay una luz amarilla apuntando en tal y tal dirección y hay una luz café rojizo en cualquier lugar [OPEN07]. En general, primero se especifica el color o el esquema de color y luego se dibuja los objetos. El color actual se mantiene hasta que este sea cambiado.

Por ejemplo, este pseudocódigo:

```
color_actual(rojo);
dibujar_objeto(A);
dibujar_objeto(B);
color_actual (verde);
color_actual (azul);
dibujar_objeto(C);
```

dibuja los objetos A y B en rojo, y el objeto azul será C. El comando que pone como color actual el verde es desperdiciado y no afecta ningún objeto consecutivo.

Para definir un color se usa el comando glColor3f(). Este toma tres parámetros (rojo, verde, azul), todos como números de punto flotante entre 0.0 y 1.0.

Por ejemplo,

```
glColor3f(1.0, 0.0, 0.0);
```

define el rojo mas brillante que el sistema pueda dibujar sin componentes verde ni azul. Todos los valores en 0 crean negro y todos los valores en 1 crean blanco.

```
glColor3f(0.0, 0.0, 0.0); negro
glColor3f(1.0, 0.0, 0.0); rojo
glColor3f(0.0, 1.0, 0.0); verde
glColor3f(1.0, 1.0, 0.0); amarillo
glColor3f(0.0, 0.0, 1.0); azul
glColor3f(1.0, 0.0, 1.0); magenta
glColor3f(0.0, 1.0, 1.0); turquesa
glColor3f(1.0, 1.0, 1.0); blanco
```

El comando anterior glClearColor(), toma cuatro parámetros, los primeros tres coinciden con los parámetros de glColor3f(). El cuarto parámetro es el valor de alfa, que se cubrirá en detalle en secciones próximas, el cual es 0.0 por defecto.

2.6.1.2 Forzando que el Dibujado se Complete

Como se revisó en el pipeline, los modernos sistemas gráficos son como una línea de ensamblaje. En tal arquitectura, no hay necesidad de que el CPU espere que cada comando de dibujado se complete antes de enviar el próximo. Mientras que el CPU esta enviando vértices por el pipeline, el hardware de transformación trabaja en el último vértice enviado, el anterior a ese esta siendo recortado, y así consecutivamente [OPEN07]. En tal sistema, si el CPU esperara que cada comando se complete habría una gran penalidad en desempeño.

Adicionalmente, la aplicación pudiera estar corriendo en mas de una maquina. Por ejemplo el programa principal se ejecuta en otro lugar (en una maquina cliente) y se visualizan resultados de dibujado en otra Terminal (el servidor) la cual esta conectada por una red con el cliente.

Por estas razones, OpenGL provee un comando glFlush(), para forzar al cliente a enviar el paquete de red aunque este no este lleno todavía. Si no hay una red y todo se ejecuta en la misma Terminal, glFlush() podría no tener efecto. Sin embargo, si se desea escribir un programa que funcione bien con o sin una red, se deberá incluir glFlush() al final de cada escena. Nótese que glFlush() no espera que el dibujado se complete, este fuerza que el dibujado se ejecute,

garantizando que todos los comandos previos se ejecuten en un tiempo finito [OPEN07].

Si glFlush() no es suficiente, se utiliza glFinish(). Este comando hace lo mismo que glFlush() pero espera notificación del hardware grafico o red indicando que el dibujado esta completo en el framebuffer.

2.6.2 Los Sistemas de Coordenadas

Cuando inicialmente se abre una ventana o posteriormente se la mueve o redimensiona, el sistema de ventanas envía un evento para notificar estas acciones. Al utilizar GLUT (véase GLUT: OpenGL Utility Library), esta notificación es automática y se llamara cualquier rutina registrada en el método glutReshapeFunc(). Por ahora solo es necesario saber que esta función reestablecerá la región rectangular en donde se realiza el renderizado y definirá el sistema de coordenadas en el cual los objetos serán dibujados [OPEN07].

Ejemplo 3: Registrar la función glutReshapeFunc(reshape) para el redimensionamiento de la ventana.

```
void reshape (int w, int h)
{
  glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  glMatrixMode (GL_PROJECTION);
  glLoadIdentity ();
  gluOrtho2D (0.0, (GLdouble) w, 0.0, (GLdouble) h);
}
```

Internamente GLUT envía a esta función los argumentos del ancho y alto de la nueva ventana en píxeles.

2.6.2.1 Descripción de Puntos, Líneas y Polígonos

Los términos de punto, línea y polígono son similares en OpenGL a sus definiciones matemáticas, pero no son las mismas.

Por ejemplo, en un monitor de computadora, la unidad desplegable más pequeña es el píxel, aunque este es 1/100 de pulgada es mucho más grande que la definición matemática de “infinitamente pequeño” de un punto.

2.6.2.1.1 Puntos

Un punto es representado por un conjunto de valores de punto flotante llamado vértice.

2.6.2.1.2 Líneas

En OpenGL, el termino línea se refiere a un segmento de línea y no al concepto matemático de extenderse hacia el infinito en ambas direcciones. Las líneas son representadas por los vértices de sus extremos.

2.6.2.1.3 Polígonos

Polígonos son las áreas encerradas por un bucle de segmentos de línea, donde los segmentos de línea son especificados por los vértices de sus extremos. Los polígonos pueden ser rellenados, delineados o ser solo puntos.

Generalmente los polígonos son complicados, así que OpenGL hace algunas restricciones sobre estos, por ejemplo, los extremos de un polígono no pueden interceptarse (polígono simple) y estos deben ser convexos[42]. Sin embargo OpenGL no restringe el número de segmentos de un polígono. Polígonos con huecos no pueden ser descritos. La razón de estas restricciones es para proveer un renderizado más rápido de los polígonos simples.

[42] Se dice que una región es convexa si para cada par de puntos de esta, el segmento que los une está totalmente incluido en la misma región.

2.6.2.1.4 Curvas y Superficies Curvas

Cualquier línea curva o superficie puede ser aproximada a cualquier grado de exactitud con pequeños segmentos de línea o pequeños polígonos. Por ejemplo al subdividir de manera que cada subsegmento sea menor al de un píxel.

Ilustración 96: Aproximación de superficies [OPEN07].

2.6.2.1.5 Especificación de Vértices

Para especificar un vértice se usa el comando glVertex*(). Sólo se puede llamar a la función glVertex*() dentro del par glBegin() y glEnd().

Ejemplo 4: Usos correctos de glVertex*().

```
glVertex2s(2, 3);
glVertex3d(0.0, 0.0, 3.1415926535898);
glVertex4f(2.3, 1.0, -2.2, 2.0);
GLdouble dvect[3] = {5.0, 9.0, 1992.0};
glVertex3dv(dvect);
```

2.6.3 Dibujado de Primitivas Geométricas en OpenGL

Para dibujar o definir líneas, puntos o polígonos, cada grupo de vértices debe estar dentro del par glBegin() y glEnd() que indica a OpenGL donde empieza y termina una primitiva gráfica. El argumento de glBegin() determina el tipo de primitiva a dibujar a partir de los vértices propiciados.

Ejemplo 5: Un polígono relleno.

```
glBegin(GL_POLYGON);
  glVertex2f(0.0, 0.0);
  glVertex2f(0.0, 3.0);
  glVertex2f(4.0, 3.0);
  glVertex2f(6.0, 1.5);
  glVertex2f(4.0, 0.0);
glEnd();
```

Tabla 19: Parámetros permitidos para glBegin() [OPEN07].

Parámetro	Descripción
GL_POINTS	Puntos individuales
GL_LINES	Par de vértices se interpreta como segmento de línea
GL_POLYGON	Los vértices describen el contorno de un polígono
GL_TRIANGLES	Cada triplete de vértices se interpreta como un triángulo
GL_QUADS	Cada cuarteto de vértices se interpreta como un cuadrilátero
GL_LINE_STRIP	Serie de líneas conectadas
GL_LINE_LOOP	Serie de líneas conectadas, con unión entre el primer y último vértice
GL_TRIANGLE_STRIP	Cada nuevo vértice se interpreta como un triángulo entre los dos anteriores vértices y el nuevo
GL_TRIANGLE_FAN	Se dibujan triángulos con un vértice común
GL_QUAD_STRIP	Igual que TRIANGLE_STRIP pero con cuadriláteros

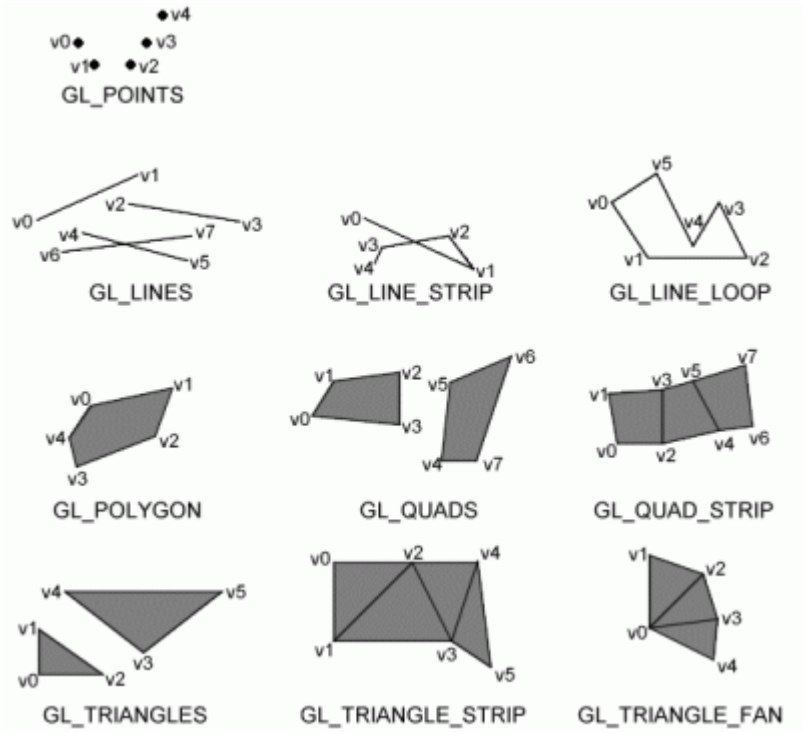


Ilustración 97: Tipos de primitivas geométricas [OPEN07].

2.6.4 Manejo Básico de los Estados

Como se revisó anteriormente, una de las variables de estado de OpenGL es el color. OpenGL mantiene muchas variables y muchos estados. Un objeto puede ser renderizado con luz, textura, niebla u otros estados que afectan su apariencia. Por defecto, estos estados están inactivos ya que algunos de ellos tienen un costo de desempeño asociado.

Para encender o apagar estos estados se usa los comandos:

```
void glEnable(GLenum cap);
void glDisable(GLenum cap);
```

glEnable() enciende un estado o capacidad, y glDisable() lo deshabilita. Hay como 40 valores enumerados que pueden ser pasados como parámetro a glEnable() o glDisable(). Algunos de ellos son GL_BLEND, GL_DEPTH_TEST, GL_FOG, GL_LINE_STIPPLE, GL_LIGHTING, etc [OPEN07].

También es posible verificar si un estado esta activo o no.

```
GLboolean glIsEnabled(GLenum capability)
```

Este método retorna GL_TRUE o GL_FALSE dependiendo de si ese estado esta activado. Sin embargo algunos valores de estado son más que solo encendido y apagado, por ejemplo el color, así que para conocer el valor de dicho estado se utiliza los siguientes comandos:

```
void glGetBooleanv(GLenum pname, GLboolean *params);
void glGetIntegerv(GLenum pname, GLint *params);
void glGetFloatv(GLenum pname, GLfloat *params);
void glGetDoublev(GLenum pname, GLdouble *params);
void glGetPointerv(GLenum pname, GLvoid **params);
```

Así se obtiene valores booleanos, enteros, decimales, de doble precisión o punteros. Una lista completa de los valores posibles para *pname* puede encontrarse en [OPEN07].

Adicionalmente se puede manipular el tamaño de los puntos, el ancho de las líneas, inclusive seguir patrones para el relleno de los polígonos a través de estados y variables dentro de OpenGL. Por ejemplo, para cambiar el tamaño de un punto se utiliza el método glLineWidth(). Para crear líneas hechas de puntos o segmentadas, se usa el comando glLineStipple() para definir el patrón a seguir y luego se activa GL_LINE_STIPPLE con glEnable().

```
glLineStipple(1, 0x3F07);
glEnable(GL_LINE_STIPPLE);
```

Un polígono tiene dos caras, frontal y posterior, y pueden ser renderizadas de maneras diferentes. Para hacer esto se usa el comando:

```
void glPolygonMode(GLenum face, GLenum mode);
```

El parámetro *face* puede ser GL_FRONT_AND_BACK, GL_FRONT, o GL_BACK; *mode* puede ser GL_POINT, GL_LINE, o GL_FILL para indicar que se debe dibujar el polígono como puntos, líneas o relleno.
Por defecto ambas caras de un polígono son rellenas y se usa un patrón sólido, pero también es posible usar un patrón distinto [OPEN07].

```
void glPolygonStipple(const GLubyte *mask);
```

Este método define el patrón actual de relleno. El argumento es una mascara (mapa de bits de 32x32) donde 1 indica que se debe dibujar y 0 que se debe dejar vacío. Esta opción es activada o desactivada por glEnable() o glDisable() con el argumento GL_POLYGON_STIPPLE.

2.6.5 Vectores Normales

Un vector normal o una normal es un vector que apunta en una dirección que es perpendicular a una superficie. Para una superficie plana, la dirección perpendicular es la misma en todos los puntos de esta, pero para una superficie curva, la normal puede ser diferente en cada punto. Con OpenGL, se puede especificar vectores normales para cada polígono o por cada vértice. No se puede asignar normales en ningún otro lado que no sea un vértice.

Los vectores normales de una superficie definen la orientación de esta en el espacio, en especial, la orientación con relación a una o varias fuentes de luz.

El comando glNormal*() asigna la normal actual al vértice especificado con glVertex*().

Ejemplo 6: Vectores normales en los vértices de la superficie de un polígono.

```
glBegin (GL_POLYGON);
  glNormal3fv(n0);
  glVertex3fv(v0);
  glNormal3fv(n1);
  glVertex3fv(v1);
```

```
  glNormal3fv(n2);
  glVertex3fv(v2);
  glNormal3fv(n3);
  glVertex3fv(v3);
glEnd();
```

2.6.5.1 Calculando un Vector Normal

Un vector normal puede ser fácilmente calculado para cualquier polígono al tomar 3 puntos que estén en el plano del polígono. La Ilustración 98 muestra tres puntos (P1, P2, y P3) que pueden ser usados para definir dos vectores: el vector V1 desde P1 hasta P2, y el vector V2 desde P1 hasta P3. Matemáticamente, dos vectores en un espacio tridimensional definen un plano. Si se realiza un producto cruz [PAN96] o producto vectorial[43] de estos vectores (V1 X V2), de manera que el vector resultante es perpendicular a ese plano.

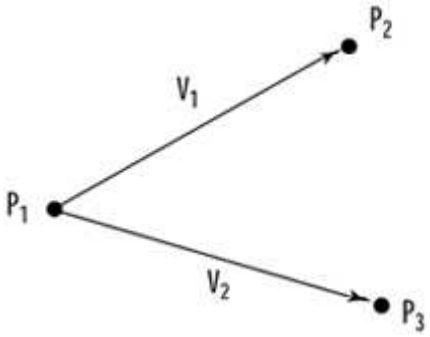


Ilustración 98: Dos vectores definidos por 3 puntos en un plano [WRIG04].

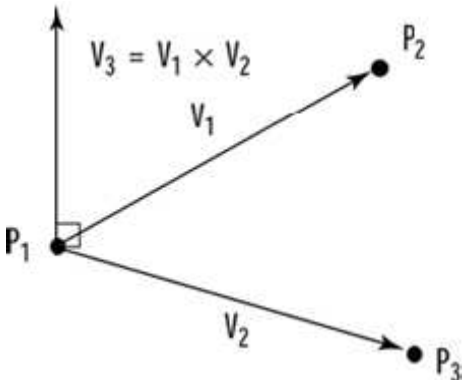


Ilustración 99: Un vector normal como el producto cruz de dos vectores [WRIG04].

[43] En álgebra lineal, el producto vectorial es una operación binaria entre dos vectores de un espacio euclídeo tridimensional que da como resultado un vector ortogonal a los dos vectores originales

2.7 Vistas

Los procesos necesarios para producir la vista deseada de una escena en 3D son similares a los procesos necesarios para tomar una fotografía. Esto es llamado la analogía de la cámara [OPEN07].

Por ejemplo se seguirían los siguientes pasos:

1. Apuntar el trípode y apuntar la cámara hacia la escena.

2. Acomodar la escena (los objetos involucrados en la fotografía) de la manera deseada.

3. Escoger los lentes para la cámara o ajustar el zoom.

4. Determinar que tan grande se desea que sea la fotografía final y se toma la fotografía.

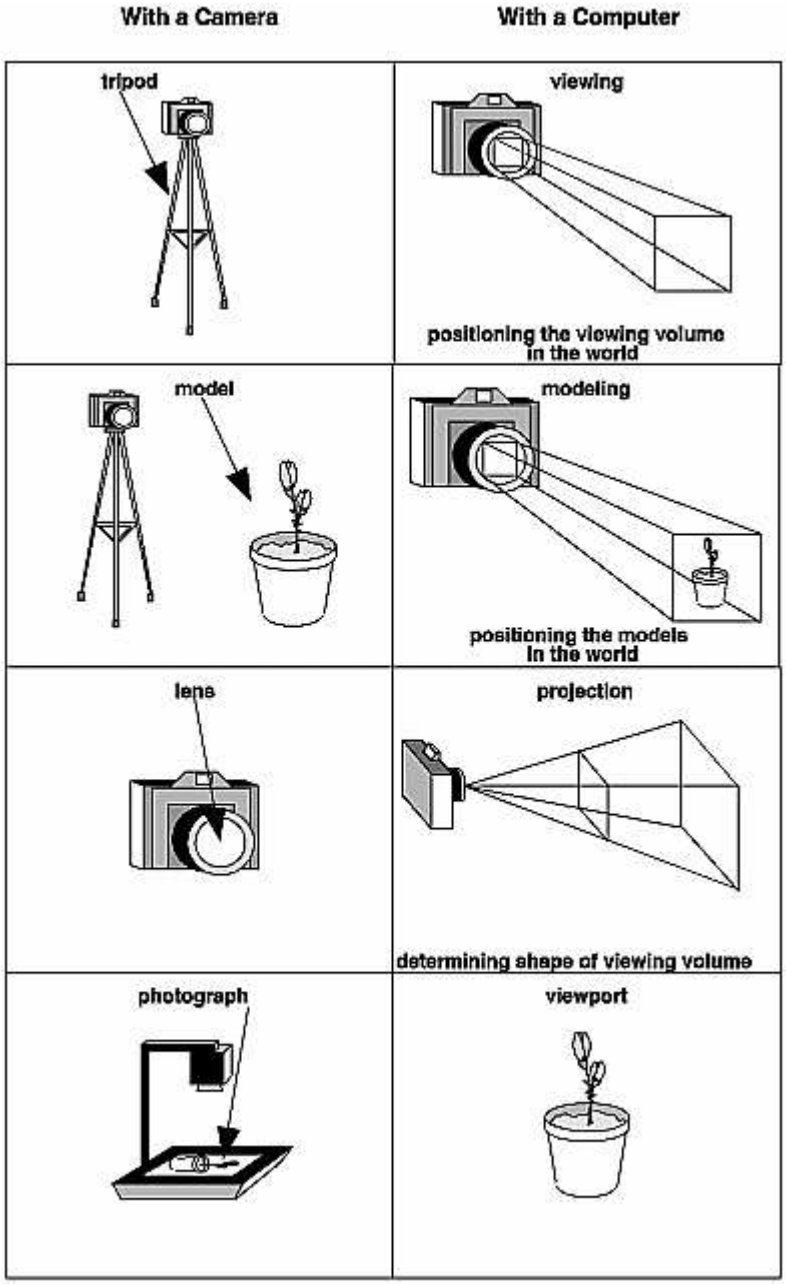


Ilustración 100: La analogia de la camara [OPEN07].

Al especificar transformaciones (rotación, traslación, escalado) en OpenGL, se realizan operaciones de multiplicación de matrices cuadradas de 4x4, ya que los vértices se componen de 4 valores (x, y, z, w) donde usualmente w es 1 y en gráficos bidimensionales z es 0. Estas matrices son declaradas como un array (array[16]) por conveniencia para las operaciones y tienen un formato de orden de columna.

$$M = \begin{bmatrix} m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \\ m_4 & m_8 & m_{12} & m_{16} \end{bmatrix}$$

Ilustración 101: Matriz en orden de columna [OPEN07]

Tomando esto en cuenta, es de suponer que las operaciones escritas en código OpenGL son ejecutadas en orden inverso como se explica más ampliamente en [OPEN07].

2.7.1 Trasformaciones de Vista y de Modelos

Las transformaciones de vista o visión de la escena y de los modelos involucrados, están íntimamente relacionadas ya que utilizan la misma matriz (GL_MODELVIEW). Dentro de las transformaciones disponibles tenemos la traslación gltranslate*(), la rotación glRotate*() y el escalado glScale*().

Para ejecutar las diferentes transformaciones, es necesario primero especificar que matriz se va a modificar, por ejemplo la matriz de vista de modelos GL_MODELVIEW o la matriz de proyección GL_PROJECTION. Esto se realiza con el comando glMatrixMode() que toma como parámetro el tipo de matriz a utilizar.

También es posible ejecutar manualmente una transformación al realizar multiplicaciones de matrices con comandos como glLoadMatrix*() o glMultMatrix*() [OPEN07], pero usualmente los comandos de OpenGL como glRotate*() ya realizan estas operaciones de matrices y a menudo de una manera más optimizada y veloz.

En OpenGL el orden de las operaciones es crucial para el resultado deseado debido a que la multiplicación de matrices no es conmutativa, como se muestra a continuación:

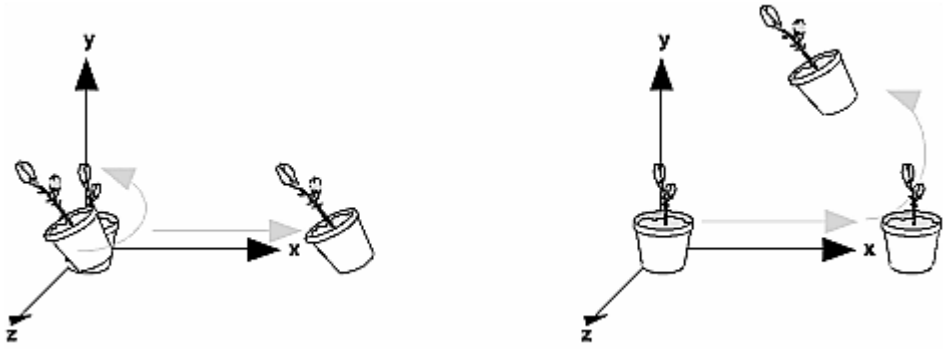


Ilustración 102: Rotar y trasladar (izq.), Trasladar y luego rotar (der.) [OPEN07].

También hay que mencionar que la transformación de escalado debe realizarse con prudencia, ya que esta operación puede influir en el rendimiento de la aplicación y además puede distorsionar un objeto irremediablemente como se explica en [OPEN07].

Ejemplo 7: Transformaciones a un cubo.

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glTranslatef(0.0f, 0.0f, -10.0f);
glutSolidCube(1.0f);
```

El método glLoadIdentity() utilizado en el ejemplo anterior, carga una matriz identidad[44] en la matriz de vista de modelos (o cualquier otra matriz definida), es decir, limpia la matriz o la inicializa para poder realizar cualquier transformación requerida.

$$\begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}$$

Ilustración 103: Matriz identidad

[44] En álgebra lineal, la matriz identidad es una matriz que cumple la propiedad de ser el elemento neutro del producto de matrices

Las transformaciones de vista o visión son análogas a mover la cámara en vez de mover los objetos en si. Es por esto que si se desea acomodar una escena, se puede escoger entre mover el objeto o mover la cámara, tomando en cuenta el sentido inverso en cada ocasión.

Para hacer una transformación de visión, también se puede utilizar el método gluLookAt() que encapsula las transformaciones requeridas para apuntar la cámara en cierta dirección.

Ejemplo 8: Apuntar la cámara hacia una posición específica.

```
gluLookAt(4.0, 2.0, 1.0, 2.0, 4.0, -3.0, 2.0, 2.0, -1.0);
```

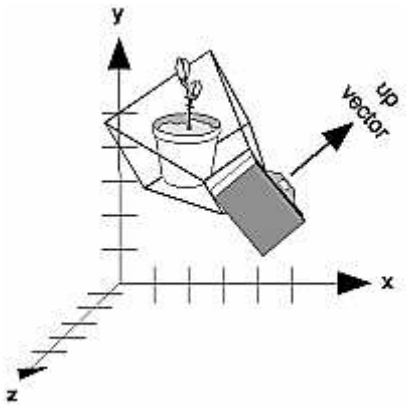


Ilustración 104: Ajuste de la cámara con glLookAt() [OPEN07]

Por defecto la cámara descansará en el origen, (0,0,0) apuntando hacia el eje Z negativo, y estará perpendicular al eje Y.

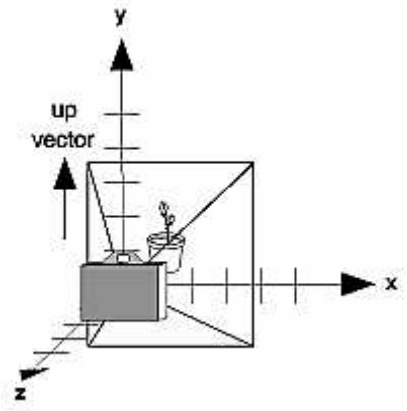


Ilustración 105: Posición por defecto de la cámara [OPEN07].

2.7.2 Transformaciones de Proyección

Para manipular transformaciones de proyección es necesario especificar la matriz de proyección GL_PROJECTION:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
```

El propósito de las transformaciones de proyección es definir el volumen de visión que se tendrá de la escena.

2.7.2.1 Proyección en Perspectiva

Como se revisó anteriormente, la característica más reconocida de los gráficos 3D es la perspectiva, con la cual los objetos distantes se ven más pequeños que los objetos más cercanos [WRIG04].

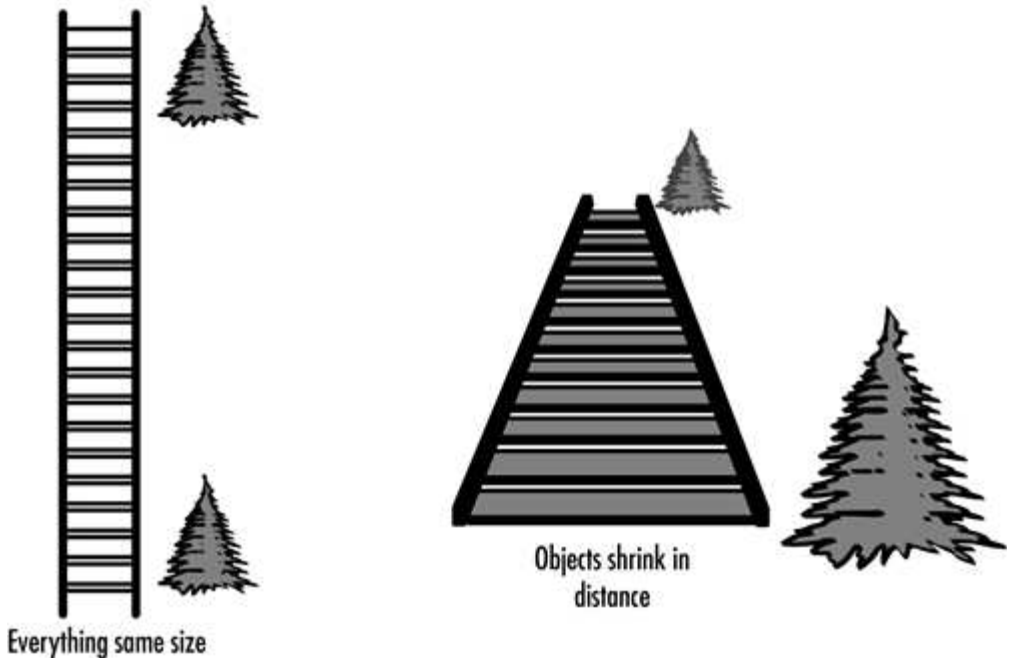


Ilustración 106: Proyeccion ortografica (izq.), proyeccion en perspectiva (der.) [WRIG04].

Este efecto se obtiene al definir un área de visión parecida a una pirámide recortada su punta conocida como frustum. El comando para definir este modo de proyección es glFrustum():

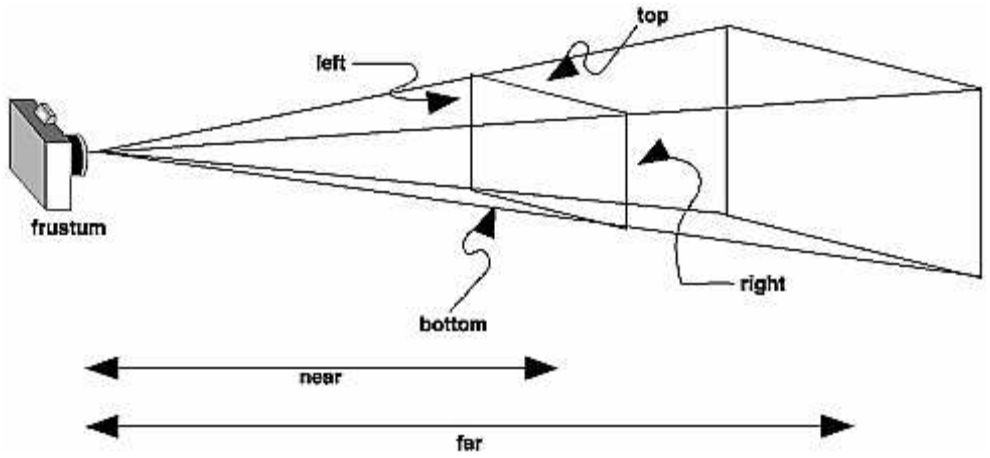


Ilustración 107: Volumen de proyección en perspectiva con el comando glFrustum() [OPEN07]

A veces glFrustum() es un poco complicado de aplicar, así que el comando gluPerspective() es utilizado en su lugar.

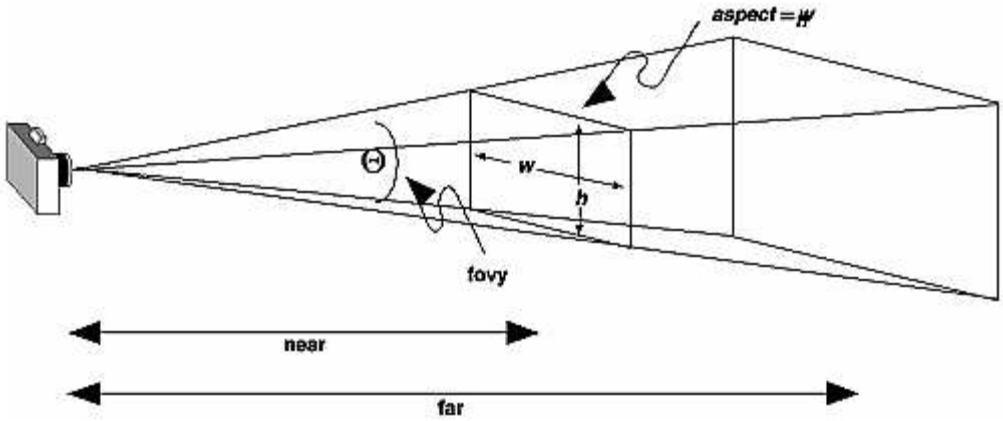


Ilustración 108: Volumen de proyección en perspectiva con el comando gluPerspective() [OPEN07]

2.7.2.2 Proyecciones Ortográficas

Como anteriormente mencionado, en una proyección ortográfica, el volumen de visión es rectangular y consistente como en un paralelepípedo[45]. Este tipo de proyección no utiliza perspectiva alguna y los objetos lejanos y cercanos no tendrán variación su tamaño actual. Este tipo de proyección es muy usada en

[45] Un paralelepípedo es un poliedro de seis caras, cada una de las cuales es un paralelogramo, que son paralelas e iguales dos a dos.

diseño arquitectónico donde es crucial mantener el tamaño real de los objetos que son proyectados.

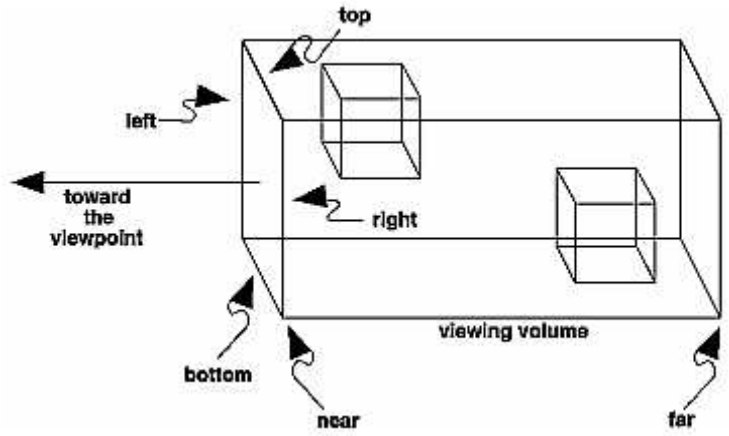


Ilustración 109: Volumen de la proyección ortográfica [OPEN07].

El método utilizado para definir este tipo de proyección es glOrtho(), aunque también esta disponible el método gluOrtho2D(), que realiza lo mismo que glOrtho() pero con la excepción de que los objetos son considerados con su coordenada z entre los valores -1 y 1, aunque usualmente al trabajar con proyecciones 2D el valor de z será de 0 [OPEN07].

Es importante aclarar que los volúmenes de proyección definen automáticamente 6 planos de recorte (clipping planes). Esto quiere decir que cualquier objeto o parte de un objeto que este fuera de estos planos será recortado para ocupar únicamente el volumen que se ha establecido.

2.7.3 Transformación del Puerto de Vista

Recordando la analogía de la cámara, esta transformación corresponde a la etapa donde se escoje el tamaño de la fotografía. El gestor de ventanas del sistema, no OpenGL, es el responsable de crear una ventana en la pantalla y por defecto, el viewport ocupara toda la ventana creada.

El comando glViewport() define el área de dibujado, que puede ser inclusive mas pequeño que la ventana o subdividir esta en 2 o mas viewports. El aspect ratio[46] del viewport generalmente seria igual al aspect ratio del volumen de visión. Si las dos relaciones son diferentes (veáse Ilustración 110), la imagen mostrada se vería distorsionada [OPEN07].

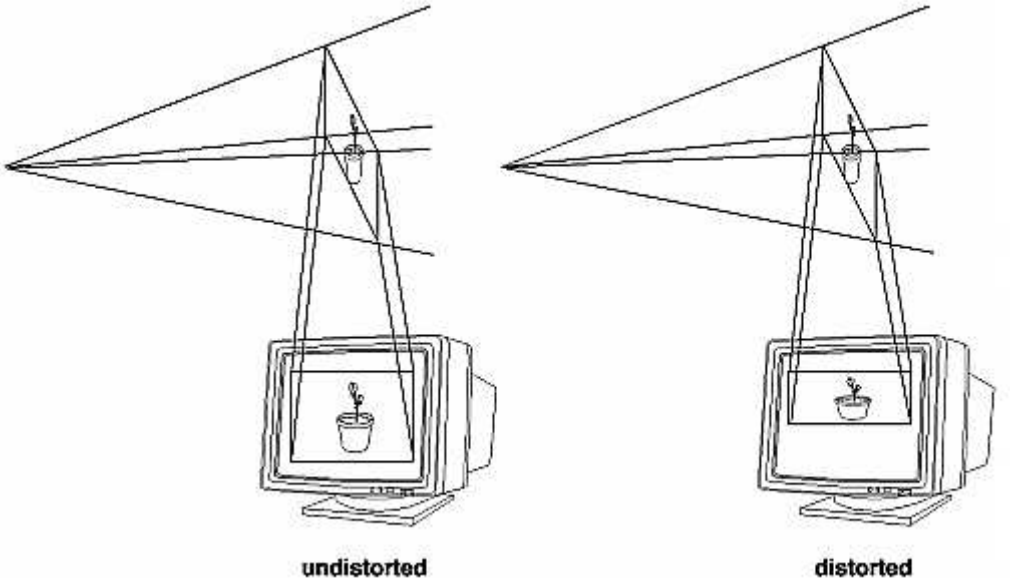


Ilustración 110: Relación entre volumen de visión y el viewport [OPEN07].

2.7.4 Las Pilas de Matrices

Como se pudo observar anteriormente, las matrices de transformación deber ser limpiadas o reiniciadas con una matriz identidad antes de realizar una operación que modifique a un objeto. Este método no siempre es el más deseado ya que a menudo se desea guardar el estado actual de la transformación y restaurarlo luego de colocar otros objetos. Este método es más conveniente cuando se ha transformado la matriz de vista y esta ya no esta en el origen.

Para facilitar este procedimiento, OpenGL mantiene una pila[47] de matrices para visión y proyección (Ilustración 111). De manera que una matriz puede ser

[46] Es la relación ancho/alto de una pantalla o una película.
[47] Una pila (stack) es una estructura de datos de tipo LIFO (Last In First Out, último en entrar, primero en salir) que permite almacenar y recuperar datos.

empujada (push) en la pila para luego sacarla (pop) y restaurar su estado anterior, tal como en una pila normal [OPEN07].

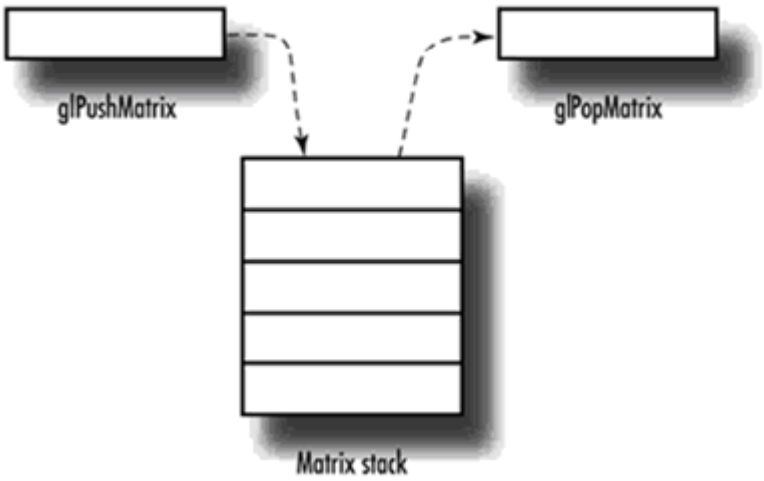


Ilustración 111: La pila de matrices [WRIG04].

Los comandos usados para la manipulación de la pila son glPushMatrix() y glPopMatrix(). El tamaño de la pila es dependiente de la implementación, por ejemplo, la implementación de Microsoft mantiene 32 niveles para la matriz de visión/modelos y 2 para la matriz de proyección.

Ejemplo 9: Un Sistema Planetario

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
static int year = 0, day = 0;
void init(void)
{
glClearColor (0.0, 0.0, 0.0, 0.0);
glShadeModel (GL_FLAT);
}
void display(void)
{
glClear (GL_COLOR_BUFFER_BIT);
glColor3f (1.0, 1.0, 1.0);
glPushMatrix();
  glutWireSphere(1.0, 20, 16); /* el Sol */
  glRotatef ((GLfloat) year, 0.0, 1.0, 0.0);
```

```
  glTranslatef (2.0, 0.0, 0.0);
  glRotatef ((GLfloat) day, 0.0, 1.0, 0.0);
  glutWireSphere(0.2, 10, 8); /* un planeta */
glPopMatrix();
glutSwapBuffers();
}
void reshape (int w, int h)
{
glViewport (0, 0, (GLsizei) w, (GLsizei) h);
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
gluPerspective(60.0, (GLfloat) w/(GLfloat) h, 1.0, 20.0);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt (0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
}
void keyboard (unsigned char key, int x, int y)
{
switch (key) {
  case `d':
    day = (day + 10) % 360;
    glutPostRedisplay();
    break;
  case `D':
    day = (day - 10) % 360;
    glutPostRedisplay();
    break;
  case `y':
    year = (year + 5) % 360;
    glutPostRedisplay();
    break;
  case `Y':
    year = (year - 5) % 360;
    glutPostRedisplay();
    break;
  default:
    break;
}
}
```

```
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100, 100);
glutCreateWindow (argv[0]);
init ();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc(keyboard);
glutMainLoop();
return 0;
}
```

2.8 Color

2.8.1 Generación de Fotones en el Computador

La pantalla del computador o monitor, genera intensidades separadas de componentes de luz roja, verde y azul, cada una con intensidades variables. En resumen, el monitor es un “arma” de electrones que los dispara hacia la parte posterior de la pantalla. Esta pantalla contiene fósforos que emiten el color rojo, verde o azul al ser estimulados por los electrones [WRIG04]. La intensidad de la luz emitida depende de la intensidad del rayo de electrones. Los fósforos se encuentran bien empaquetados entre si de manera que representen un punto físico en la pantalla.

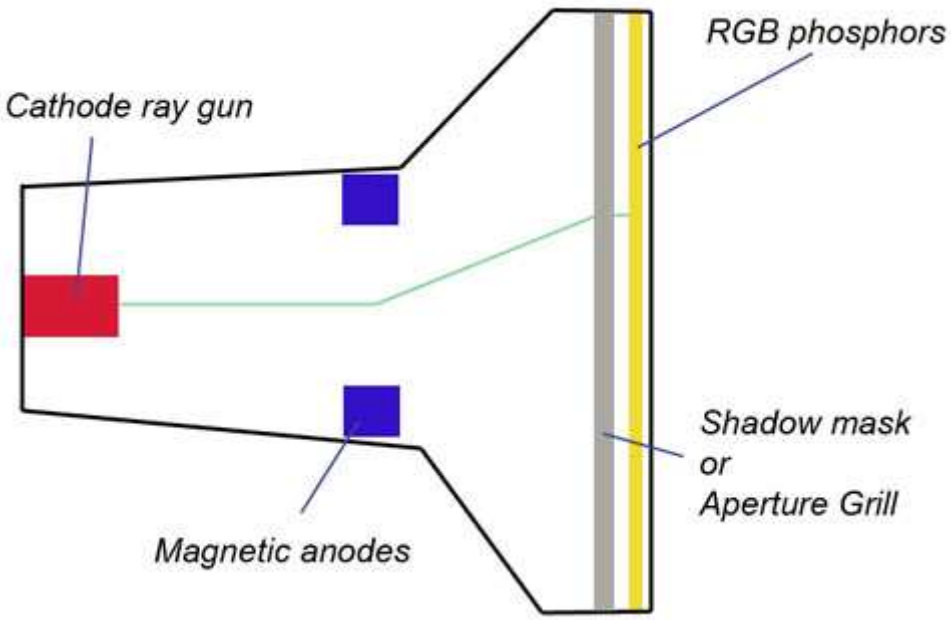


Ilustración 112: Disección de un monitor de computadora.

Las pantallas de cristal liquido LCD trabajan de una manera un poco distinta, estos utilizan cristal liquido para permitir, o nó, el paso de la luz de fondo hacia los subpixeles (un conjunto de rojo, verde, azul) y así crear el color [WEB03].

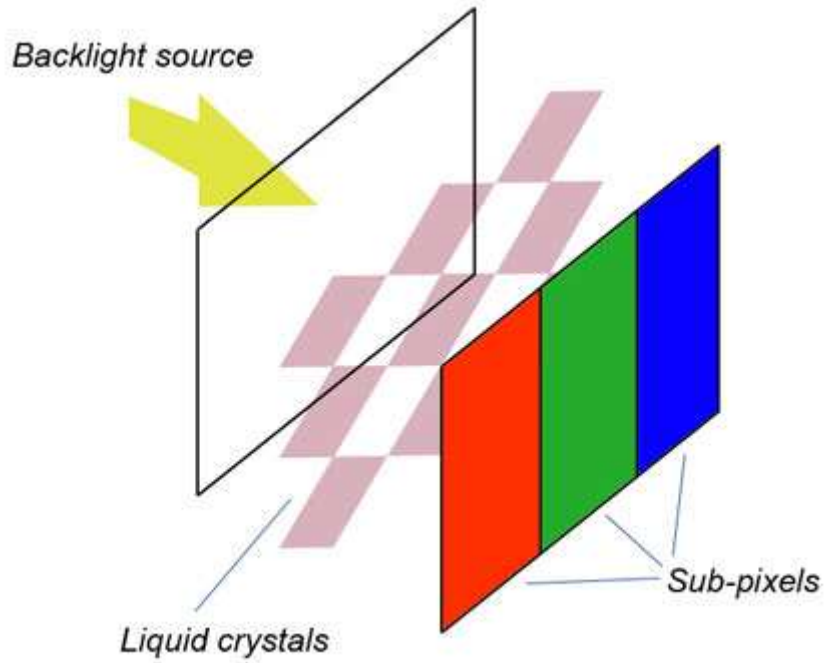


Ilustración 113: Sección de una pantalla LCD.

2.8.2 Color en OpenGL

Microsoft Windows y Apple Macintosh revolucionaron el mundo de los gráficos por computador al crear los primeros sistemas con entornos gráficos que fueron muy bien aceptados por las empresas y el consumidor. Además, hicieron que los gráficos por computador sean más fáciles de manejar por los programadores. Esto gracias a la virtualización de hardware, es decir la creación de controladores (drivers). De manera que no era necesario escribir instrucciones directamente al hardware grafico, ya que los programadores disponían de una interfaz de programación (API) tal como lo es OpenGL [WRIG04].

2.8.2.1 Modo de Color RGBA

Como ya se revisó anteriormente (véase Color y Colorimetría), sabemos que el color es especificado por tres valores positivos de rojo, verde y azul (modo RGB o RGBA), de manera que se puede representar este modelo como un espacio de color.

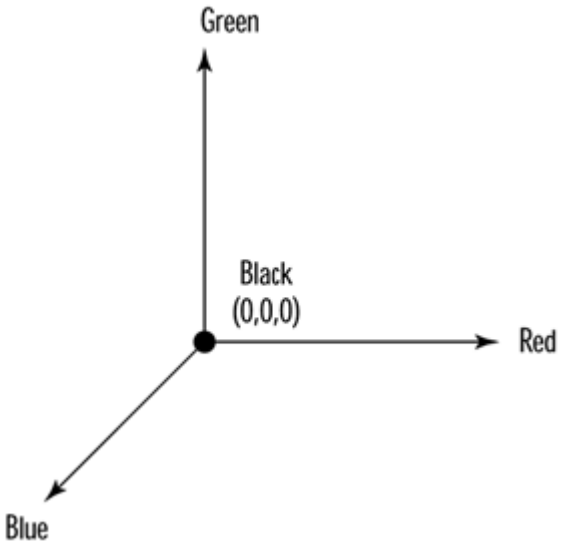


Ilustración 114: El espacio de color RBG [WRIG04].

Cada coordenada de este espacio corresponde a un color determinado, por ejemplo, el origen corresponde al color negro (0,0,0) es decir la falta o ausencia de color y el eje X corresponde al componente rojo.

Dependiendo de la capacidad de almacenamiento de la memoria de video, por ejemplo el modo de 24 bits [OPEN07], se destinarán 8 bits para cada componente: 8 para rojo, 8 para verde y 8 para azul. En el modo de 32 bits también llamado “color real”, se maneja los bits de manera idéntica, pero se agrega 8 bits para manejar el componente alfa [WRIG04]. Un valor de 255 en algún componente significa saturación de ese valor, es decir, el máximo nivel de ese color que dicho monitor puede representar.

Ahora, para representar todos los colores posibles, se extiende la dimensionalidad del espacio de color hacia el cubo de color (véase Ilustración 115). Por ejemplo, toda la gama de grises o tonos de gris se definen en la diagonal desde (0,0,0) hasta (255,255,255). La gama de un color o su sombreado es muy importante ya que se utiliza mucho para simular la iluminación de un objeto al obtener un color intermedio o degradado que cubre su superficie [WRIG04].

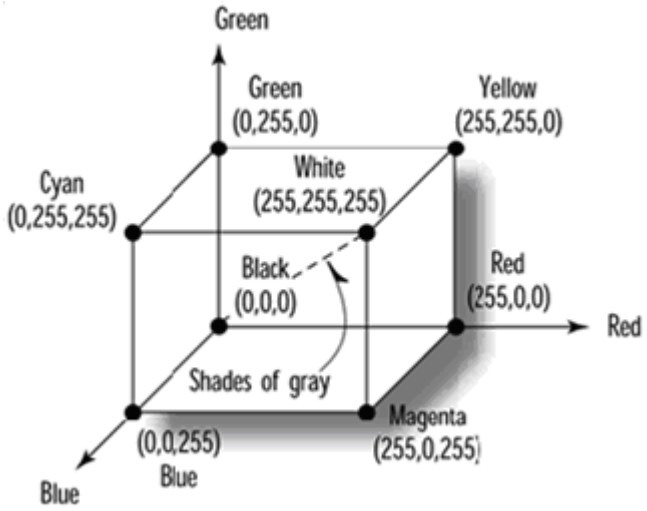


Ilustración 115: El cubo de color RGB [WRIG04].

Para definir un color en OpenGL, se utiliza el metodo glColor*() y se pasa como parámetro los componentes RGB o RGBA si se esta trabajando como este modo de color.

Ejemplo 10: Representar un punto rojo

```
glColor3f (1.0, 0.0, 0.0); // rojo al nivel máximo posible
glBegin (GL_POINTS);
  glVertex3fv (point_array);
glEnd ();
```

El comando glColor*() asigna el color actual que va a ser usado para todos los vértices que serán dibujados a continuación. Pero si se especifica diferentes colores para los vértices de un mismo objeto, el resultado dependerá del tipo de sombreado (shading) que se este utilizando [WRIG04]. Por ejemplo, si una línea, que se compone de 2 vértices, a cada vértice se le asigna un color distinto, de manera que a lo largo de la línea, se obtiene una interpolación de color de cada vértice. En el caso de los polígonos, se interpola el color del interior en base a sus vértices.

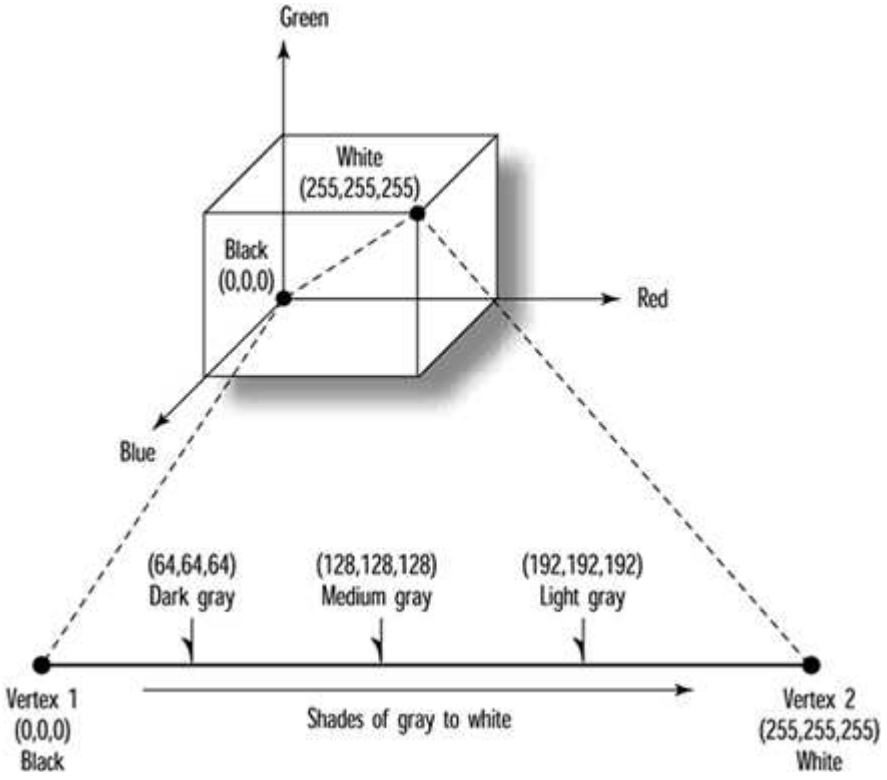


Ilustración 116: Sombreado de una línea desde negro hacia blanco [WRIG04].

En OpenGL tenemos dos tipos de sombreado, GL_FLAT y GL_SMOOTH. El primero es un sombreado plano de manera que el color asignado a un vértice determina el color de la primitiva completa, en cambio que el segundo tipo de sombreado utiliza la técnica de Gouraud[48] [WEB01] la cual realiza la interpolación de color de cada uno de los vértices que componen la primitiva.

Para asignar el tipo de modelo de sombreado en OpenGL se utiliza el comando glShadeModel() y se pasa como parámetro GL_FLAT o GL_SMOOTH.

Ejemplo 11: Un triangulo con sombreado Gouraud

```
glShadeModel(GL_SMOOTH);
glBegin(GL_TRIANGLES);
  glColor3ub((GLubyte)255,(GLubyte)0,(GLubyte)0);
  glVertex3f(0.0f,200.0f,0.0f);
  glColor3ub((GLubyte)0,(GLubyte)255,(GLubyte)0);
```

[48] Henri Gouraud (1944) es un científico francés inventor del sombreado que lleva su nombre (Gouraud shading) usado ampliamente en gráficos por computadora.

```
  glVertex3f(200.0f,-70.0f,0.0f);
  glColor3ub((GLubyte)0,(GLubyte)0,(GLubyte)255);
  glVertex3f(-200.0f, -70.0f, 0.0f);
glEnd();
```

2.8.2.2 Modo de Color Indexado

El modo de color indexado en OpenGL se basa en una tabla de valores o mapa de colores [OPEN07] la cual es similar a usar una paleta de colores, la cual esta indexada para permitir la mezcla de las tonalidades de rojo, verde y azul.

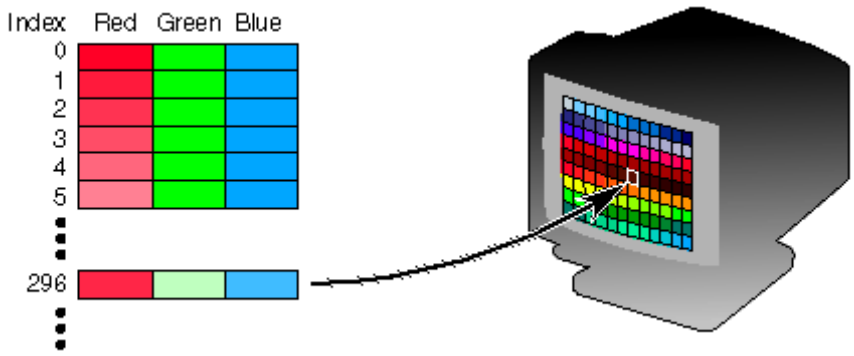


Ilustración 117: El mapa de colores del Modo Indexado.

En este modo de color, el número de colores disponibles simultáneos es limitado por el tamaño del mapa de colores y el número de planos de bits (bitplanes) disponibles. El tamaño del mapa de colores es determinado por la cantidad de hardware dedicado a este y generalmente va desde el rango 256^{28} hasta 4096^{212}, donde el exponente es el número de bitplanes que se están usando [OPEN07].

La decisión entre usar el modo indexado o el modo RGBA dependerá mucho del tipo de hardware disponible y el fin de la aplicación. En la mayoría de sistemas, el modo RGBA representa más colores de manera simultánea que el modo indexado. Además, muchos efectos como sombreado, iluminación, texturización y niebla son fáciles de implementar en modo RGBA. Así que la decisión entre un modo u otro depende de cómo esta orientada la aplicación y en que sistemas se ejecutará.

2.9 Iluminación en OpenGL

OpenGL aproxima la iluminación al separar la luz en los componentes rojo, verde y azul (véase La Colorimetría Tricromática). De manera que el color de las fuentes de luz se caracteriza por la cantidad de rojo, verde y azul que emiten, y el material de las superficies es caracterizado por el porcentaje de rojo, verde y azul que es reflejado en varias direcciones. Las ecuaciones que OpenGL utiliza para la iluminación solo son una aproximación, pero realiza un muy buen trabajo y puede ser calculada de una manera relativamente veloz [OPEN07]. Si se requiere un modelo de iluminación más exacto o simplemente diferente, habría que realizar nuestros propios cálculos, y dicho código seria enormemente complejo y talvez no tan eficiente en un sistema de tiempo real.

2.9.1 El modelo de Iluminación de OpenGL

En el modelo de iluminación de OpenGL, la luz en una escena proviene de varias fuentes de luz que pueden ser individualmente encendidas o apagadas. Algunas luces provienen de una dirección en particular, en cambio otras están esparcidas en la escena. Por ejemplo, al encender el foco de una habitación, la mayoría de la luz proviene del bulbo, pero una parte de la iluminación también proviene de la luz que ha rebotado de las paredes [OPEN07]. Esta luz que rebota es llamada la luz ambiente (ambient) o ambiental y se asume que se esparce por toda la escena de manera que no hay una forma segura de decir cual es su dirección original.

En OpenGL, las fuentes de luz solo tienen efecto si la superficie afectada tiene propiedades de absorción o reflexión de la luz. Cada superficie es considerada como un conjunto de propiedades llamado *material* [OPEN07]. Un material puede emitir su propia luz, esparcir la luz en todas direcciones o reflejar parte de la luz incidente en una dirección especifica como si fuese un espejo u otra superficie brillante.

El modelo de iluminación de OpenGL considera la iluminación como una división de 4 componentes independientes: emisivo, ambiente, difuso y especular. Todos estos componentes son computarizados de manera independiente y luego unidos entre si.

2.9.1.1 Luz Ambiente en OpenGL

El componente ambiente no proviene de una dirección particular. Tiene una fuente, pero los rayos de luz han rebotado en la escena y se han convertido en rayos sin dirección. Los objetos iluminados por este tipo de luz son alumbrados de manera equilibrada en todas las superficies y en todas las direcciones [WRIG04].

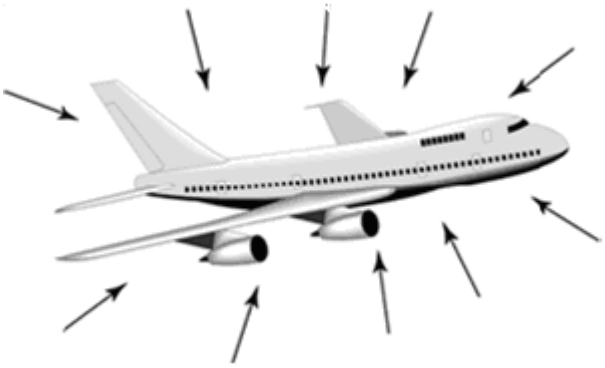


Ilustración 118: Un objeto iluminado sólo por luz ambiente [WRIG04].

2.9.1.2 Luz Difusa en OpenGL

La luz difusa (véase Color y Colorimetría: Campo físico) proviene de una dirección particular pero es reflejada de manera equitativa en toda la superficie. Aunque la luz es reflejada de manera equilibrada, la superficie del objeto es más brillante si la luz es apuntada sobre el objeto más que si esta incide desde un ángulo [WRIG04].

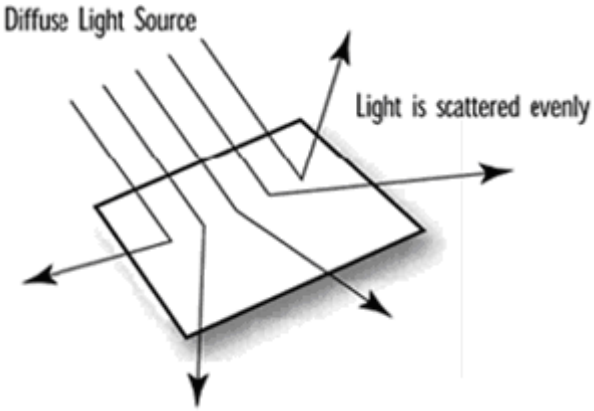


Ilustración 119: Un objeto iluminado por una fuente de luz difusa [WRIG04].

2.9.1.3 Luz Especular en OpenGL

La luz especular (veáse Color y Colorimetría: Campo físico) proviene de una dirección en especial y es reflejada como en un espejo en una dirección particular. El metal brillante o el plástico tienen un componente especular muy alto, mientras que una alfombra tiene un componente especular de casi 0 [OPEN07]. Se puede pensar en luz especular como en el brillo de una superficie, ya que esta luz crea un punto brillante en una superficie, el cual es llamado brillo especular[49].

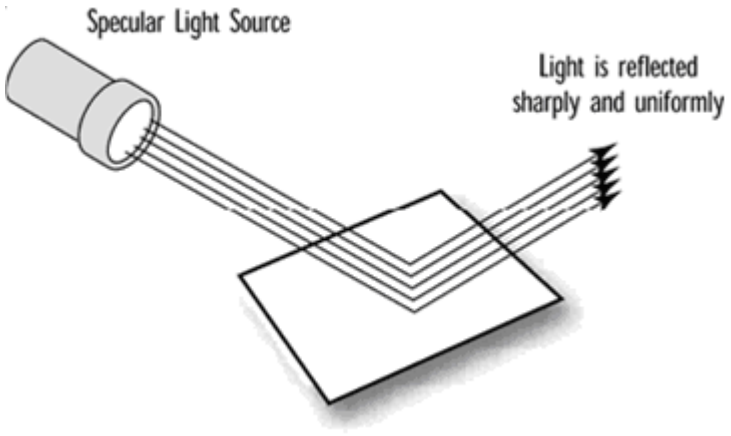


Ilustración 120: Un objeto iluminado por una fuente de luz especular.

Ninguna fuente de luz es compuesta enteramente de un solo componente, sino es una combinación de las intensidades de luz ambient, difusse y specular. Por

[49] El brillo especular (specular highlight) es un punto brillante de luz que aparece en objetos muy pulidos o brillantes cuando son iluminados.

ejemplo, un rayo láser esta compuesto de un componente puro especular rojo. Sin embargo, el humo, polvo u otras partículas que están en el aire afectan al láser, de manera que puede ser visto a través de la habitación. Esta dispersión representa el componente difuso del láser. Si el láser es brillante y ninguna otra luz esta presente en la escena, los objetos en la habitación tomarán un tono rojizo. Esto es la presencia de un muy bajo componente ambiente [WRIG04].

Entonces, una fuente de luz en una escena se dice esta compuesta de tres componentes de iluminación: ambient, diffuse, y specular. Estos componentes son definidos de igual manera que un color en OpenGL, con los valores RGBA que describen las intensidades de rojo, verde, azul y alfa.

Tabla 20: Valores de los componentes de luz de un láser rojo.

	Rojo	Verde	Azul	Alfa
Specular	0.99	0.0	0.0	1.0
Diffuse	0.10	0.0	0.0	1.0
Ambient	0.05	0.0	0.0	1.0

2.9.2 Materiales en OpenGL

La luz es solo una parte de la ecuación. En el mundo real, los objetos tienen un color propio. Anteriormente se definió el color de un objeto por las longitudes de onda de la luz que este refleja. Por ejemplo, una pelota azul refleja y absorbe la mayoría de los fotones azules. Con esto se asume que la luz que ilumina la pelota tiene fotones azules para ser reflejados y detectados por el observador [WRIG04].

Generalmente, en la mayoría de las escenas se opta por una luz blanca que contiene una mezcla equilibrada de todos los colores [WRIG04]. Bajo una luz blanca, la mayoría de los objetos muestran su color natural, aunque esto no se cumple en ciertos casos.

2.9.2.1 Propiedades de los Materiales

Cuando se utiliza la iluminación en una escena, ya no se describen los polígonos como que tuvieran un color en particular, sino que estos consisten en materiales que tienen ciertas propiedades reflectivas. En vez de decir que un polígono es rojo, se dice que un polígono esta hecho de un material que refleja la mayoría de la luz roja. Todavía se describe que la superficie es roja, pero ahora se debe especificar también las propiedades reflectivas para los componentes ambient, diffuse, y specular. Un material puede ser brillante y reflejar la luz especular muy bien, mientras que otro puede absorber toda la luz especular y nunca lucir brillante [WRIG04].

Otra propiedad es la de emisión para simular que los objetos emitan su propia luz agregándoles un toque de intensidad. Este tipo de propiedad no agrega luz alguna en la escena [OPEN07].

Definir los valores de la iluminación y las propiedades de los materiales para obtener un efecto deseado toma mucha práctica y experiencia. No existen reglas específicas para esto, es así que este es el punto en que la ciencia se convierte en magia. La manera en que OpenGL logra estos efectos es básicamente tomando los valores de color de los componentes de cada vértice (ambient, difusse y specular) y multiplicándolos por los componentes de la luz, incorporando el sombreado o degradado entre vértices [WRIG04].

2.9.2.2 Creación de Fuentes de Luz

Las fuentes de luz tienen propiedades específicas, algunas de ellas son el color, la posición y la dirección hacia donde apuntan. Adicionalmente existen otras propiedades para ciertos tipos de luz. Por ahora es importante saber que en OpenGL se especifica las propiedades de una fuente de luz con el comando glLight*() el cual toma tres parámetros: el identificador de la luz a la cual se le asignara o modificara sus propiedades, la propiedad en si, y el valor deseado para dicha propiedad [OPEN07].

Tabla 21: Valores por defecto para el primer parámetro de glLight*() [OPEN07]

Parámetro	Valor por defecto	Significado
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	Intensidad ambiental RGBA de la luz
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	Intensidad difusa RGBA de la luz
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	Intensidad especular RGBA de la luz
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	Posición de la luz (x, y, z, w)
GL_SPOT_DIRECTION	(0.0, 0.0, -1.0)	Dirección de la luz (x, y, z)
GL_SPOT_EXPONENT	0.0	Exponente del foco
GL_SPOT_CUTOFF	180.0	Angulo de apertura del foco
GL_CONSTANT_ATTENUATION	1.0	Factor de atenuación constante
GL_LINEAR_ATTENUATION	0.0	Factor de atenuación lineal
GL_QUADRATIC_ATTENUATION	0.0	Factor de atenuación cuadrático

Ejemplo 12: Definición de posición y colores para una fuente de luz

```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

2.9.2.3 Color

OpenGL permite asociar tres parámetros relacionados con el color (GL_AMBIENT, GL_DIFFUSE, y GL_SPECULAR) con cualquier fuente de luz. Por ejemplo, el parámetro GL_AMBIENT se refiere a la intensidad RGBA de la luz ambiente que cierta fuente de iluminación contribuye a la escena.

```
GLfloat light_ambient[] = { 0.0, 0.0, 1.0, 1.0};
glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
```

2.9.2.4 Posición y Atenuación

En OpenGL una fuente de luz puede ser tratada como si estuviese infinitamente lejos de la escena o como si estuviese muy cerca de la escena [OPEN07]. El primer tipo de luz es conocido como una fuente de luz direccional; el efecto que esta luz contribuye es que los rayos de luz pueden ser considerados paralelos al momento que alcanzan a un objeto. Un ejemplo de este tipo de luz es la luz solar. El segundo tipo es conocido como una fuente de luz posicional, ya que su posición exacta en la escena determina el efecto que esta tiene y la dirección de donde provienen los rayos de luz. Un ejemplo de este tipo de luz seria una lámpara de escritorio.

```
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_POSITION, light_position);
```

En las fuentes de luz reales, la intensidad de la luz decrece a medida que la distancia desde la fuente de luz se incrementa. Este efecto es conocido como atenuación [OPEN07]. Debido a que una luz direccional esta infinitamente lejos, no es necesario atenuar su intensidad respecto a su distancia, entonces la atenuación estaría deshabilitada para una luz direccional. Sin embargo, la atenuación es importante para dar realismo con luces posicionales. En OpenGL la atenuación de una fuente de luz es el producto de la contribución de esa fuente de luz por el factor de atenuación:

$$\text{factor de atenuación} = \frac{1}{k_c + k_l d + k_q d^2}$$

donde:

d = distancia entre la posición de la luz y el vértice
kc = GL_CONSTANT_ATTENUATION
kl = GL_LINEAR_ATTENUATION
kq = GL_QUADRATIC_ATTENUATION

Ejemplo 13: Definición de los valores de atenuación.

```
glLightf(GL_LIGHT0, GL_CONSTANT_ATTENUATION, 2.0);
glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 1.0);
glLightf(GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.5);
```

Los valores de las contribuciones de luz ambiente, difusa y especular son atenuados, solo el valor de emisión y luz ambiente global no son atenuados. También es importante recalcar que al involucrar cálculos matemáticos adicionales, especialmente divisiones, para cada color, el uso de luces atenuadas afecta el rendimiento de la aplicación [OPEN07].

2.9.2.5 Spotlights

Un spotlight (foco) es una fuente de luz posicional a la cual se le ha restringido la luz que emite a un cono. Para crear un spotlight es necesario determinar la amplitud del cono de iluminación deseado. Para especificar el ángulo de apertura del foco se usa el parámetro GL_SPOT_CUTOFF. El cual es duplicado para obtener el ángulo total de apertura [OPEN07].

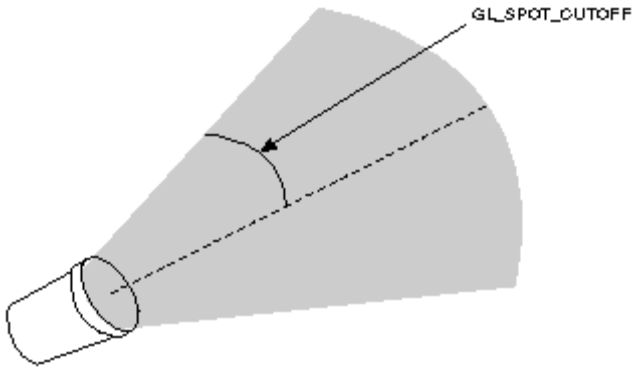


Ilustración 121: El parámetro GL_SPOT_CUTOFF de un spotlight [OPEN07]

Por defecto, esta opción esta desactivada ya que el parámetro por defecto es 180º (véase Tabla 21). El valor de GL_SPOT_CUTOFF se restringe al rango [0.0, 90.0] o el valor especial de 180 [OPEN07]. También es necesario determinar la dirección del spotlight que por defecto apunta al eje Z negativo.

Ejemplo 14: Creación de un spotlight

```
GLfloat spot_direction[] = { -1.0, -1.0, 0.0 };
glLightfv(GL_LIGHT0, GL_SPOT_DIRECTION, spot_direction);
glLightf(GL_LIGHT0, GL_SPOT_CUTOFF, 45.0);
```

Adicionalmente hay dos maneras más de controlar la intensidad de la distribución de la luz dentro del cono. Primero asignando el valor de atenuación, el cual es multiplicado por la intensidad de la luz. También se puede asignar el valor del parámetro GL_SPOT_EXPONENT, el cual controla la concentración de la luz. La intensidad de la luz es mayor en el centro del cono y atenuada hacia los bordes por el coseno del Angulo entre la dirección de la luz y la dirección desde la luz hasta el vértice que se ilumina, elevado a la potencia del exponente del foco GL_SPOT_EXPONENT. De manera que mientras mayor sea el exponente, la luz será más enfocada [OPEN07].

2.9.2.6 Luces Múltiples

En OpenGL se puede tener hasta 8 fuentes de luz (es posible tener más dependiendo de la implementación de OpenGL) [OPEN07]. Debido a que OpenGL realiza varias operaciones matemáticas para determinar cuanta luz recibe un vértice por cada fuente de luz, un número elevado de luces afectará el rendimiento de la aplicación.

Para definir el identificador de una fuente de luz, se utiliza GL_LIGHT0, GL_LIGHT1, GL_LIGHT2, GL_LIGHT3, etc.

Ejemplo 15: Dos fuentes de luz simultáneas

```
GLfloat light_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
GLfloat light1_position[] = { -2.0, 2.0, 1.0, 1.0 };
GLfloat light2_position[] = { 2.0, -2.0, 1.0, 1.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_POSITION, light1_position);
glLightfv(GL_LIGHT1, GL_AMBIENT, light_ambient);
```

```
glLightfv(GL_LIGHT1, GL_POSITION, light2_position);

glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
```

2.9.3 Selección de un Modelo de Iluminación

La noción que tiene OpenGL [OPEN07] sobre un modelo de iluminación se basa en tres componentes:

- La intensidad de la luz ambiental global.
- La posición del punto de vista es local en la escena o debería ser considerado infinitamente distante.
- Los cálculos de iluminación deberían ser realizados de manera diferente para las caras frontales y posteriores de los objetos.

El comando usado para especificar las propiedades del modelo de iluminación es glLightModel*(), el cual tiene como argumentos la propiedad del modelo de iluminación y el valor deseado para esa propiedad.

Tabla 22: Valores por defecto del primer parámetro de glLightModel*() [OPEN07]

Nombre del Parámetro	Valor por defecto	Significado
GL_LIGHT_MODEL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	Intensidad RGBA de toda la escena
GL_LIGHT_MODEL_LOCAL_VIEWER	0.0 o GL_FALSE	Como se computan los ángulos de reflexión especular
GL_LIGHT_MODEL_TWO_SIDE	0.0 o GL_FALSE	Iluminación de un solo lado o de ambos lados
GL_LIGHT_MODEL_COLOR_CONTROL	GL_SINGLE_COLOR	El color especular debería ser calculado por separado del ambiente y difuso.

2.9.3.1 Luz Ambiental Global

Como se presentó anteriormente, cada fuente de luz contribuye luz ambiental a la escena. Adicionalmente, puede existir otra luz ambiental que no proviene de una fuente en particular [OPEN07]. Para especificar los valores RGBA de dicha fuente de luz, se utiliza el parámetro GL_LIGHT_MODEL_AMBIENT:

```
GLfloat lmodel_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
```

2.9.3.2 Punto de Vista Local o Infinito

El lugar del punto de vista afecta los cálculos de los resplandores o brillos especulares en OpenGL, los cuales se producen por una reflectancia especular. De una manera mas explicita, la intensidad del resplandor en un vértice en particular depende de la normal en ese vértice, la dirección desde el vértice hacia la fuente de luz, y la dirección del vértice hacia el punto de vista [OPEN07].

Con un punto de vista infinito, la dirección entre este y cualquier vértice en la escena permanece constante. Un punto de vista local brinda mayor realismo a la escena, pero ya que la dirección tiene que ser calculada para cada vértice, el desempeño general se verá afectado. Por defecto se asume un punto de vista infinito.

```
glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
```

2.9.3.3 Iluminación en Ambos Lados

Los cálculos de iluminación se realizan para cada polígono en una escena, sean estos de cara frontal o posterior. Usualmente, se configura la iluminación para las caras frontales de los polígonos, entonces la iluminación de las caras posteriores no es siempre correctamente calculada.

Cuando se activa la opción de iluminación en ambas caras (frontal y posterior)

```
glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);
```

OpenGL revierte las normales de las superficies, lo que significa que las partes ocultas al observador se iluminaran de manera correcta, sin embargo, estas operaciones adicionales haces que la iluminación en ambas caras tengan un desempeño inferior a la iluminación de una sola cara [OPEN07].

2.9.3.4 Activar la Iluminación

Tal como se revisó previamente, la iluminación en OpenGL es una variable de estado, la cual necesita ser explícitamente activada (o desactivada). Si la iluminación no esta habilitada, el color actual es mapeado de manera simple en cada vértice, y no se realizan cálculos que involucran normales, fuentes de luz, modelos de iluminación o propiedades de materiales.

Para activar la iluminación se procedería de la siguiente manera:

```
glEnable(GL_LIGHTING);
```

También es necesario activar una a una las fuentes de luz que se agregan a la escena luego de que se hayan definido los parámetros deseados de cada una:

```
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
```

Para desactivar o deshabilitar una fuente de luz o la iluminación en general se utiliza el comando glDisable() con el parámetro adecuado.

2.9.3.5 Color especular secundario

Para cálculos típicos de iluminación, las contribuciones ambiente, difuso, especular y de emisión son calculados y simplemente sumados entre si. Por defecto, el mapeado de texturas es aplicado luego de la iluminación, así que

los destellos especulares pueden parecer opacos, o las texturas pudieran no dar los resultados deseados [OPEN07]. Para retrasar la aplicación del color especular hasta luego de la aplicación de las texturas, se hace una llamada a:

```
glLightModeli(GL_LIGHT_MODEL_COLOR_CONTROL,
GL_SEPARATE_SPECULAR_COLOR);
```

De esta manera, la iluminación produce dos colores por vértice: un primario y un secundario. El color primario se combina con los colores de la textura, y el color secundario es agregado luego del resultado de la combinación del color primario y la textura [OPEN07]. Es así que los objetos texturizados tienen brillos especulares más realistas y convincentes.

2.9.4 Definición de las Propiedades de los Materiales

Hasta ahora se ha visto como definir las propiedades del modelo de iluminación deseado y como crear fuentes de luz en una escena. La manera de definir las propiedades de los materiales de los objetos iluminados es similar, simplemente se utiliza el comando glMaterial*().

Tabla 23: Valores por defecto para pname del método glMaterial*() [OPEN07]

Nombre del Parámetro	Valor por defecto	Significado
GL_AMBIENT	(0.2, 0.2, 0.2, 1.0)	Color ambiente del material
GL_DIFFUSE	(0.8, 0.8, 0.8, 1.0)	Color difuso del material
GL_AMBIENT_AND_DIFFUSE		Color ambiente y difuso del material
GL_SPECULAR	(0.0, 0.0, 0.0, 1.0)	Color especular del material
GL_SHININESS	0.0	Exponente especular
GL_EMISSION	(0.0, 0.0, 0.0, 1.0)	Color de emisión del material
GL_COLOR_INDEXES	(0,1,1)	Índices del color ambiente, difuso y especular

2.9.4.1 Reflexión Difusa y Ambiente

Los parámetros GL_DIFFUSE y GL_AMBIENT asignados con glMaterial*() afectan el color ambiente y difuso de la luz reflejada por un objeto. El rol mas importante lo desempeña la reflectancia difusa, la cual determina el color que se percibe de un objeto. Esta es afectada por el color de la luz difusa incidente y el ángulo entre la luz que incide y la dirección de la normal, de manera que es mas intensa cuando la luz que incide cae de manera perpendicular a la superficie [OPEN07].

La reflectancia ambiental afecta el color general del objeto. Ya que la reflectancia difusa es más brillante donde un objeto es iluminado, la reflectancia ambiente es más perceptible cuando un objeto no recibe iluminación directa. La reflectancia ambiental total de un objeto es afectada por la luz ambiental global de cada una de las luces individuales involucradas en la escena. Las reflectancia ambiental y difusa no son afectadas por la posición del punto de vista [OPEN07].

En objetos del mundo real, las reflectancias difusa y ambiente tienen normalmente el mismo color. Por esta razón, OpenGL provee una manera conveniente de asignar ambos valores simultáneamente:

```
GLfloat mat_amb_diff[] = { 0.1, 0.5, 0.8, 1.0 };
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE,mat_amb_diff);
```

2.9.4.2 Reflexión Especular

La reflexión especular en un objeto produce un resplandor o brillo en un punto del mismo. Diferente de la reflexión ambiente y difusa, la cantidad de reflexión especular vista por el observador depende del lugar del punto de vista, es decir, es más brillante a través del ángulo directo de reflexión [OPEN07]. Por ejemplo, al observar una esfera metálica en el exterior en un día soleado, al mover nuestra cabeza, el destello creado parece seguirnos hasta cierto punto. Sin embargo si nos movemos mucho perderemos la visión de la reflexión.

Con OpenGL se puede determinar el efecto que tiene un material con la luz reflectada (GL_SPECULAR) y se puede controlar el tamaño y brillantez del destello (GL_SHININESS). De manera que un valor alto en GL_SHININESS (0.0-128.0) producirá un brillo pequeño y más brillante, es decir más enfocado [OPEN07].

Ejemplo 16: Definición del brillo especular

```
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat low_shininess[] = { 5.0 };
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
glMaterialfv(GL_FRONT, GL_SHININESS, low_shininess);
```

2.9.4.3 Emisión

Cuando se especifica el color RGBA del parámetro GL_EMISSION, se puede obtener el efecto de que un objeto determinado parezca producir su propia luz de dicho color [OPEN07]. Ya que la mayoría de los objetos, excepto luces, no emiten luz, este efecto se utilizaría para simular lámparas y otras fuentes de luz en una escena.

Ejemplo 17: Definición de emisión de un material

```
GLfloat mat_emission[] = {0.3, 0.2, 0.2, 0.0};
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);
```

Los objetos asignados el valor de emisión, parecen irradiar luz, pero en realidad no contribuyen con luz alguna en la escena. Sin embargo es posible colocar una fuente de luz en la misma posición del objeto para obtener ese efecto [OPEN07].

2.9.4.4 Cambiando las Propiedades de los Materiales

En algunas situaciones, no se desea que todos los objetos en una escena tengan las mismas propiedades de material, pero, llamadas repetitivas al comando glMaterial*() tiene un costo de rendimiento asociado, lo que afectaría el rendimiento general de la aplicación.

Una técnica para reducir los costos de rendimiento asociados al cambiar las propiedades de los materiales constantemente es usando la función glColorMaterial() [OPEN07]. Este comando especifica dos valores independientes: el primero define la cara del polígono que se va a actualizar, y el segundo especifica qué propiedad de material o propiedades de dicha cara o caras se va a actualizar.

Es necesario también activar el estado GL_COLOR_MATERIAL glEnable() antes de hacer uso del método glColorMaterial(). Luego hay que cambiar el color actual con glColor*() u otras propiedades de material con glMaterial*():

Ejemplo 18: Modo de empleo de glColorMaterial()

```
glEnable(GL_COLOR_MATERIAL);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
/* ahora glColor* cambiará la reflexión difusa */
glColor3f(0.2, 0.5, 0.8);
/* dibujar objeto u objetos con estas propiedades */
glColorMaterial(GL_FRONT, GL_SPECULAR);
/* glColor* ya no afecta la reflexión difusa */
/* ahora glColor* cambiará la reflexión especular */
glColor3f(0.9, 0.0, 0.2);
/* dibujar objetos con las nuevas propiedades */
glDisable(GL_COLOR_MATERIAL);
```

2.10 Blending, Antialiasing y Fog

2.10.1 Blending

Anteriormente se dio a conocer que OpenGL coloca los valores de los colores en un colorbuffer (buffer de color) bajo condiciones normales. Los valores de profundidad de cada fragmento también se colocan en el depthbuffer (buffer de profundidad). Cuando la opción de cálculos de profundidad (GL_DEPTH_TEST) está deshabilitada, los nuevos valores de los colores simplemente sobrescriben cualquier otro valor ya presente en el buffer. Cuando los cálculos o pruebas de profundidad son activados, los nuevos fragmentos de color reemplazan solamente a aquellos fragmentos considerados cercanos al plano de recorte (clipping plane) más próximo, en vez de reemplazar cualquier fragmento previo [ASTL04].

Estas reglas no se aplican cuando la opción de blending (mezclado) es activada. Cuando blending es activado, el nuevo color que ingresa es combinado con el color que ya esta presente en el buffer. La forma en que estos colores son combinados genera una gran variedad de efectos especiales [ASTL04].

Blending es una variable de estado de OpenGL, por lo cual una llamada a la función glEnable() activa o glDisable() desactiva esta opción:

```
glEnable(GL_BLENDING);
```

2.10.1.1 Combinación de Colores

La combinación de colores es la base de lo que es el blending. Operaciones matemáticas involucran los colores de destino, es decir, los colores que ya están almacenados en el buffer de color, y los colores fuente (source), que son aquellos que están ingresando recientemente y que pueden o no interactuar con los colores de destino [OPEN07]. Los colores origen y destino, pueden tener 3 o 4 componentes (rojo, verde, azul y opcionalmente alfa)

La manera en que los colores de origen y destino son combinados cuando se activa la opción de blending (mezcla) es controlada por diferentes ecuaciones simples. Por defecto, una ecuación de mezcla seria así:

$C_f = (C_s * S) + (C_d * D)$

Donde, C_f es el color final calculado, C_s es el color origen (source), C_d es el color destino, S y D son los factores de mezcla origen y destino respectivamente. Estos factores de mezcla son asignados por el comando:

```
glBlendFunc(GLenum S, GLenum D);
```

Como se puede ver, S y D son valores enumerados y no valores numéricos específicos que se asignan directamente ya que obedecen a cálculos entre los colores origen, destino, inclusive el valor alfa (A) de cada color.

Tabla 24: Factores de mezcla (blending) [ASTL04].

Función	Factor de Mezcla RGB	Factor de Mezcla Alfa
GL_ZERO	(0,0,0)	0
GL_ONE	(1,1,1)	1
GL_SRC_COLOR	(Rs,Gs,Bs)	As
GL_ONE_MINUS_SRC_COLOR	(1,1,1) – (Rs,Gs,Bs)	1 – As
GL_DST_COLOR	(Rd,Gd,Bd)	Ad
GL_ONE_MINUS_DST_COLOR	(1,1,1) – (Rd,Gd,Bd)	1 – Ad
GL_SRC_ALPHA	(As,As,As)	As
GL_ONE_MINUS_SRC_ALPHA	(1,1,1) – (As,As,As)	1 – As
GL_DST_ALPHA	(Ad,Ad,Ad)	Ad
GL_ONE_MINUS_DST_ALPHA	(1,1,1) – (Ad,Ad,Ad)	1 – Ad
GL_CONSTANT_COLOR	(Rc,Gc,Bc)	Ac
GL_ONE_MINUS_CONSTANT_COLOR	(1,1,1) – (Rc,Gc,Bc)	1 – Ac
GL_CONSTANT_ALPHA	(Ac,Ac,Ac)	Ac
GL_ONE_MINUS_CONSTANT_ALPHA	(1,1,1) – (Ac,Ac,Ac)	1 – Ac
GL_SRC_ALPHA_SATURATE	(f,f,f)[50]	1

[50] Donde f = min(As, 1 – Ad).

La manera matemática de cómo los colores y los factores de mezcla son combinados es bastante sencilla y puede ser mas ampliamente revisada en [WRIG04].

Ejemplo 19: Utilización de blending para simular reflexiones.

```
void RenderScene(void)
{
// Clear the window with current clearing color
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glPushMatrix();
// Move light under floor to light the "reflected" world
glLightfv(GL_LIGHT0, GL_POSITION, fLightPosMirror);
glPushMatrix();
glFrontFace(GL_CW); // geometry is mirrored,
// swap orientation
glScalef(1.0f, -1.0f, 1.0f);
DrawWorld();
glFrontFace(GL_CCW);
glPopMatrix();
// Draw the ground transparently over the reflection
glDisable(GL_LIGHTING);
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
DrawGround();
glDisable(GL_BLEND);
glEnable(GL_LIGHTING);
// Restore correct lighting and draw the world correctly
glLightfv(GL_LIGHT0, GL_POSITION, fLightPos);
DrawWorld();
glPopMatrix();
// Do the buffer Swap
glutSwapBuffers();
}
```

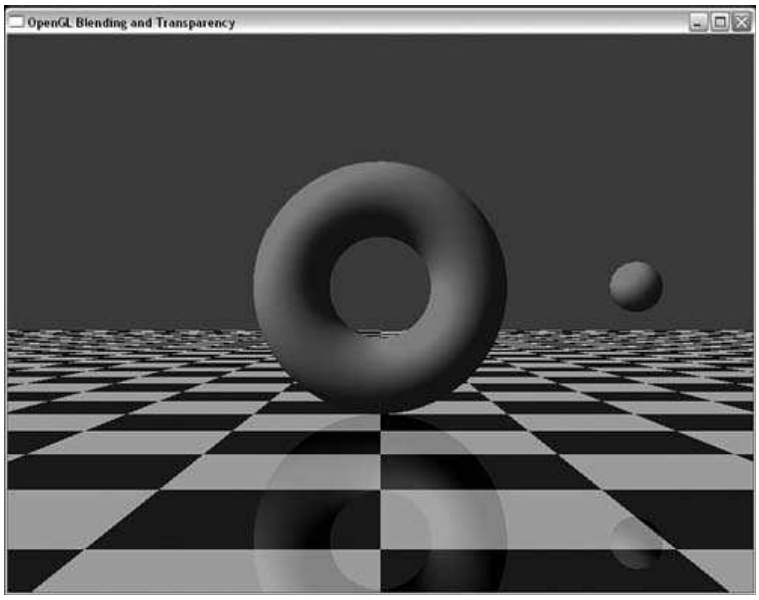


Ilustración 122: Resultado del Ejemplo 19 [WRIG04].

2.10.1.2 Cambiar la Ecuación de Mezcla

Anteriormente se mostró la ecuación por defecto que usa OpenGL:

$C_f = (C_s * S) + (C_d * D)$

Si es necesario se puede escoger de entre 5 diferentes ecuaciones usando el método glBlendEquation().

Tabla 25: Operaciones matemáticas de cada ecuación permitida [WRIG04]

Modo	Operación matematica
GL_FUNC_ADD (por defecto)	Cf = (Cs * S) + (Cd * D)
GL_FUNC_SUBTRACT	Cf = (Cs * S) – (Cd * D)
GL_FUNC_REVERSE_SUBTRACT	Cf = (Cd * D) – (Cs * S)
GL_MIN	Cf = min(Cs, Cd)
GL_MAX	Cf = max(Cs, Cd)
GL_LOGIC_OP	S op D

Adicionalmente OpenGL brinda otra función para el manejo de las ecuaciones de mezcla con el comando:

```
void glBlendFuncSeparate(GLenum srcRGB, GLenum dstRGB, GLenum
srcAlpha, GLenum dstAlpha);
```

Entonces el método glBlendFunc() especifica las funciones para los valores RGBA de origen y destino, en cambio que, glBlendFuncSeparate() permite especificar funciones de mezcla para los componentes RGB y Alfa de manera separada [WRIG04].

Adicionalemte, como se mostro en la Tabla 24, los valores GL_CONSTANT_COLOR, GL_ONE_MINUS_CONSTANT_COLOR, GL_CONSTANT_ALPHA, y GL_ONE_MINUS_CONSTANT_ALPHA permiten el uso de un color constante de mezcla para incluirlo en la ecuación correspondiente. Por defecto el color constante es negro, pero puede ser cambiado a cualquier otro color usando la función glBlendColor().

También es importante mencionar que el renderizado de objetos tridimensionales mezclados con otros usando blending puede ocasionar ciertos problemas en ciertas situaciones, ya que entra muy en cuanta la profundidad de los objetos y como estos deber ser renderizados con las reglas de eliminación de superficies ocultas. Existe una solución par estos problemas haciendo que el buffer de profundidad sea de solo lectura por un momento como se indica en [OPEN07].

2.10.2 Antialiasing

Otro de los usos de blending en OpenGL es el antialiasing (véase Artificios 3D: Antialiasing).

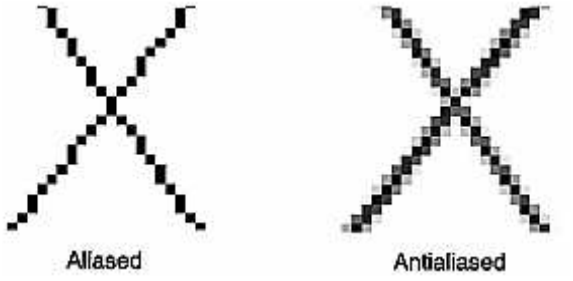


Ilustración 123: Línea normal y línea suavizada con antialiasing (der.) [OPEN07].

Para muchas tareas es necesario evitar las imágenes o gráficos con este tipo de puntiagudas o toscas líneas o polígonos. De manera que, para eliminar este efecto, OpenGL emplea el blending para mezclar los bordes de una primitiva con su entorno.

Activar esta opción es simple:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

De esta manera se puede controlar si se desea suavizar líneas, puntos o polígonos al activar sus estados:

```
glEnable(GL_POINT_SMOOTH);          // puntos suavizados
glEnable(GL_LINE_SMOOTH);           // líneas suavizadas
glEnable(GL_POLYGON_SMOOTH);        // polígonos suavizados
```

Es útil en ocasiones, usar el método glHint() para tener cierto control sobre la calidad de la imagen o del rendimiento [OPEN07].

Tabla 26: Valores permitidos para glHint() [OPEN07]

Parámetro	Significado
GL_POINT_SMOOTH_HINT, GL_LINE_SMOOTH_HINT, GL_POLYGON_SMOOTH_HINT	Especifica la calidad de muestreo de puntos, líneas o polígonos durante las operaciones de antialiasing
GL_FOG_HINT	Especifica si los cálculos de niebla (fog) deben ser por píxeles (GL_NICEST) o por vértice (GL_FASTEST)
GL_PERSPECTIVE_CORRECTION_HINT	Especifica la calidad deseada de interpolación de color y textura

El segundo parámetro de glHint() puede ser GL_FASTEST para indicar que se desea la opción más eficiente, GL_NICEST para indicar la más alta calidad, o GL_DONT_CARE para no indicar preferencia alguna.

2.10.2.1 Multisampling

A pesar de que el suavizado de primitivas es una gran ayuda, tiene su desventaja, al ser no muy soportada en todas las plataformas, en especial GL_POLYGON_SMOOTH. Aun cuando antialiasing sea soportado no es conveniente tener toda la escena suavizada, ya que se basa en operaciones de blending y puede ocasionar problemas como se menciono previamente. Pero, una adición a OpenGL 1.3 que soluciona muchos inconvenientes esta disponible, y se la conoce como multisampling (muestreo múltiple) [OPEN07].

Esta característica incluye otro buffer adicional al framebuffer que incluye valores de color, profundidad y stencil. Todas las primitivas son muestreadas varias veces por píxel y los resultados se almacenan en este buffer. Naturalmente, esta memoria extra y procesamiento incurre en penalidades de desempeño de la aplicación, inclusive algunas implementaciones de OpenGL pueden no soportar multisampling [ASTL04].

Para hacer uso de la opción de muestreo múltiple (multisampling), hay que seguir 3 pasos básicos:

1. Obtener una ventana que soporte multisampling, por ejemplo con GLUT se haria lo siguiente:
   ```
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_MULTISAMPLE);
   ```

2. Verificar si multisampling esta disponible en la implementación de OpenGL que se este usando, es decir, que el valor de GL_SAMPLE_BUFFERS sea al menos de 1 y el valor de GL_SAMPLES sea mayor que 1.
   ```
   GLint bufs, simples;
   glGetIntegerv(GL_SAMPLE_BUFFERS, &bufs);
   glGetIntegerv(GL_SAMPLES, &samples);
   ```

3. Activar la opción:
   ```
   glEnable(GL_MULTISAMPLE);
   ```

2.10.3 Fog

Las imágenes por computadora a veces lucen irrealmente claras y muy bien definidas. Fog (niebla) agrega el efecto de que los objetos desaparecen con la distancia. Fog (véase Artificios 3D: Niebla) es un término general que describe efectos atmosféricos, por ejemplo, polvo, neblina, polución, etc. Fog es esencial en aplicaciones de simulación visual, donde la visibilidad limitada debe ser aproximada.

Cuando se activa la niebla (fog), OpenGL mezcla un color de niebla especificado luego de que todos los cálculos de color se hayan completado. La cantidad de color mezclado con la primitiva varía con la distancia de la primitiva hacia la camara [ASTL04].

Es importante decir que la niebla puede incrementar el desempeño de aplicaciones grandes de simulación, ya que se puede escoger en no dibujar los objetos que estarían completamente cubiertos por niebla y no serían visibles.

La manera de incluir niebla en una aplicación es bastante simple. Se activa esta opción con glEnable(), y se escoje el color y la ecuación que controla la densidad de la niebla con glFog*(). Si se desea se puede variar el valor de GL_FOG_HINT con glHint().

Tabla 27: Ecuaciones de niebla soportadas por OpenGL [OPEN07]

Modo de Niebla	Ecuación
GL_LINEAR	f = (end – c) / (end – start)
GL_EXP	f = exp(–d * c)
GL_EXP2	f = exp(–(d * c)2)

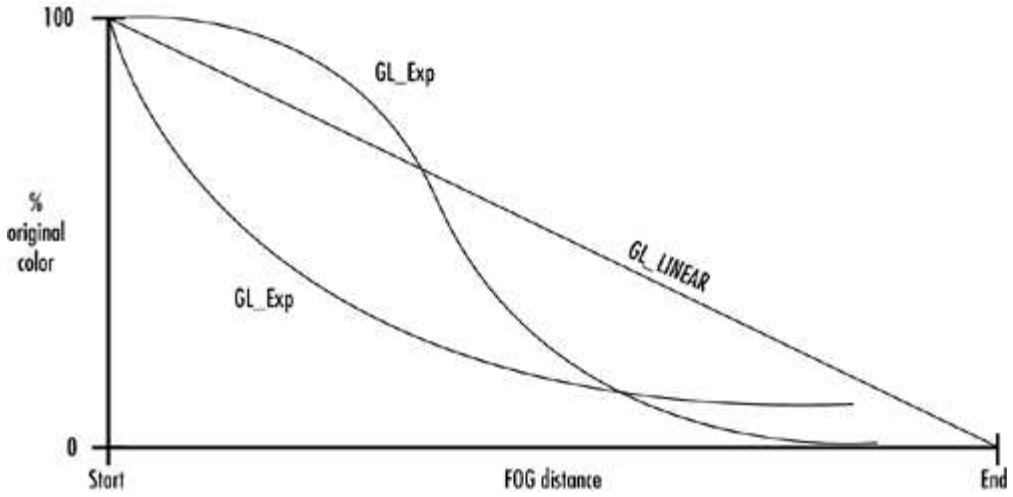


Ilustración 124: Las ecuaciones de niebla (fog) en forma gráfica [WRIG04].

En las ecuaciones previas, *c* es la distancia entre el fragmento y el punto de vista, *end* es la distancia GL_FOG_END de finalizacion, y *start* es la distancia GL_FOG_START de inicio. El valor *d* es la densidad de la niebla.

Ejemplo 20: Configuración de niebla.

```
glClearColor(0, 0, 0);
glEnable(GL_FOG); // activar la niebla
glFogfv(GL_FOG_COLOR, (0, 0, 0)); // el color de la niebla
glFogf(GL_FOG_START, 5.0f); // la distancia de inicio
glFogf(GL_FOG_END, 30.0f); // cuan lejos la niebla termina
glFogi(GL_FOG_MODE, GL_LINEAR); // ecuación de control
glFogf(GL_FOG_DENSITY, 0.5f); // densidad de la niebla
```

2.11 Mapas de Bits, Fuentes e Imágenes

Hasta ahora, es conocida la manera de renderizar datos de geometrías (puntos, líneas y polígonos). Además, OpenGL nos da la facilidad de no solo renderizar objetos vectoriales, sino tambien, objetos raster (productos de una rasterizacion) como por ejemplo los mapas de bits, típicamente usados para obtener fuentes (texto), e imágenes, las cuales pueden ser obtenidas desde los varios formatos (tga, bmp, png, etc) [OPEN07].

Los bitmaps (mapas de bits) y las imágenes de manera similar, toman la forma de un array rectangular de píxeles. La diferencia entre ambos es que el mapa de bits consiste de un solo bit de información sobre cada píxel, y las imágenes típicamente incluyen varias piezas de datos por cada píxel (los componentes RGBA por ejemplo) [ASTL04].

2.11.1 Mapas de Bits y Fuentes

Un bitmap (mapa de bits) en OpenGL es un array rectangular compuesto de 0s y 1s. De manera que un cero significa que ese bit no es tomado en cuenta y no sera dibujado, un 1 indica que se debe dibujar ese píxel con el color actual [ASTL04]. Como se mencionó, el uso mas común de los bitmaps es la obtención de caracteres en pantalla.

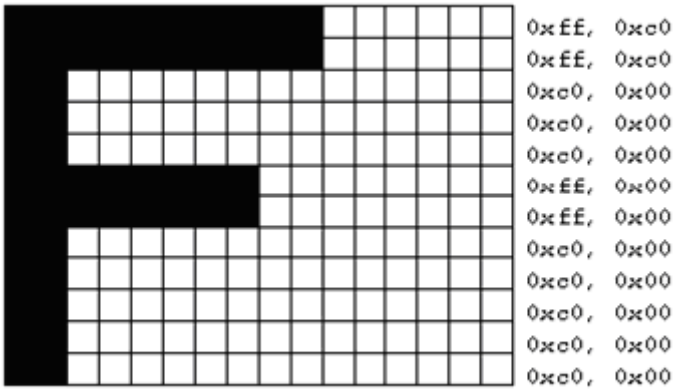


Ilustración 125: Un bitmap de la letra F [OPEN07].

La posición actual de rasterización, es el origen desde la cual el bitmap o la imagen serán dibujadas. El comando usado para definir esta posición es glRasterPos*():

```
glRasterPos2i(20, 20);
```

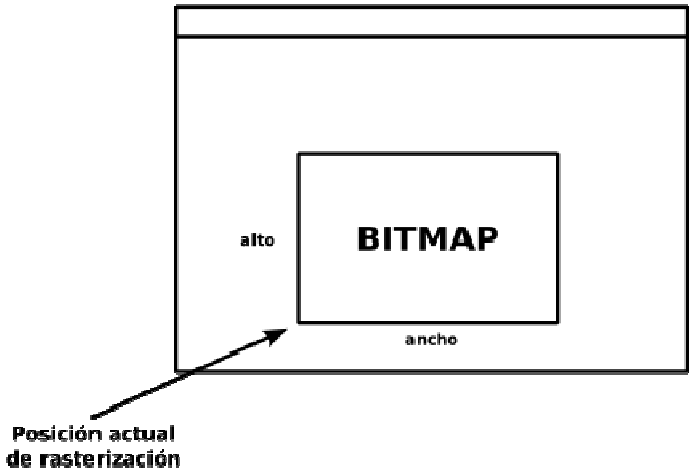


Ilustración 126: La ubicacion de glRasterPos*() respecto al bitmap.

Una vez especificada la poción inicial del bitmap, se utiliza glBitmap() para dibujar el mapa de bits en pantalla:

```
glBitmap (10, 12, 0.0, 0.0, 11.0, 0.0, bitmap);
```

Ejemplo 21: Método para renderizar varias letras A en posiciones aleatorias.

```
unsigned char letterA[] = {
            0xC0, 0x03, 0xC0, 0x03, 0xC0, 0x03,
            0xC0, 0x03, 0xC0, 0x03, 0xDF, 0xFB,
            0x7F, 0xFE, 0x60, 0x06, 0x30, 0x0C,
            0x30, 0x0C, 0x18, 0x18, 0x18, 0x18,
            0x0C, 0x30, 0x0C, 0x30, 0x07, 0xE0,
            0x07, 0xE0 };

void Render(){
      // clear screen and depth buffer
      glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
      // load the identity matrix (clear to default position and
orientation)
      glLoadIdentity();

      // single byte alignment
      glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

      // color white
      glColor3f(1.0, 1.0, 1.0);

      // render all the bitmaps
      for (int idx = 0; idx < MAX_BITMAPS; idx++)
      {
            glRasterPos2i(m_bitmaps[idx].xPos, m_bitmaps[idx].yPos);
            glBitmap(16, 16, 0.0, 0.0, 0.0, 0.0, letterA);
      }
}
```

2.11.2 Imágenes

Una imagen es similar a un mapa de bits (bitmap), pero contiene mucha más información sobre cada píxel que la conforma. Una imagen se puede obtener de varias fuentes, como por ejemplo, una fotografía digitalizada con un scanner o una imagen generada por un programa de diseño digital (Gimp, Inkscape, Blender, etc). Estas imágenes pueden ser manipuladas por OpenGL (expandidas, escaladas, volteadas) y son mayormente usadas para dar más realismo a una escena con la técnica de mapeado de texturas [OPEN07]. Debido a que OpenGL puede manipular imágenes píxel por píxel, a veces las imágenes también son conocidas como pixmaps (mapas de pixeles).

OpenGL provee tres comandos básicos para manipular datos de imagen:

1. glReadPixels() Lee un array rectangular de pixeles desde el framebuffer y los guarda en memoria del procesador.

2. glDrawPixels() Escribe un array rectangular de pixeles de los datos almacenados en la memoria del procesador en el framebuffer en una posición ráster especificada con glRasterPos*().

3. glCopyPixels() Copia un array rectangular de pixeles de una parte del framebuffer a otra. Es similar a realizar llamadas a glReadPixels() y luego a glDrawPixels().

Tabla 28: Formatos de píxeles necesarios como parámetro para glReadPixels() o glDrawPixels() [ASTL04].

Constante	Formato de Pixeles
GL_COLOR_INDEX	Color único indexado
GL_RGB	Secuencia de componentes RGB
GL_RGBA	Secuencia de componentes RGBA
GL_RED	Componente único de color rojo
GL_GREEN	Componente único de color verde
GL_BLUE	Componente único de color azul
GL_ALPHA	Componente único de color alfa
GL_LUMINANCE	Componente único de luminancia
GL_LUMINANCE_ALPHA	Componente de luminancia seguido de un componente alfa
GL_STENCIL_INDEX	Índice stencil único
GL_DEPTH_COMPONENT	Componente único de profundidad

Tabla 29: Tipos de Datos usados por glReadPixels() o glDrawPixels() [WRIG04]

Constante	Tipo de Dato
GL_UNSIGNED_BYTE	Entero positivo de 8 bits
GL_BYTE	Entero de 8 bits
GL_BITMAP	Bits unicos en enteros positivos de 8 bits usando el mismo formato que glBitmap()
GL_UNSIGNED_SHORT	Entero positivo de 16 bits
GL_SHORT	Entero de 16 bits
GL_UNSIGNED_INT	Entero positivo de 32 bits
GL_INT	Entero de 32 bits
GL_FLOAT	Punto flotante de precisión simple
GL_UNSIGNED_BYTE_3_2_2	Valores RGB empacados
GL_UNSIGNED_BYTE_2_3_3_REV	Valores RGB empacados

GL_UNSIGNED_SHORT_5_6_5	Valores RGB empacados
GL_UNSIGNED_SHORT_5_6_5_REV	Valores RGB empacados
GL_UNSIGNED_SHORT_4_4_4_4	Valores RGBA empacados
GL_UNSIGNED_SHORT_4_4_4_4_REV	Valores RGBA empacados
GL_UNSIGNED_SHORT_5_5_5_1	Valores RGBA empacados
GL_UNSIGNED_SHORT_1_5_5_5_REV	Valores RGBA empacados
GL_UNSIGNED_INT_8_8_8_8	Valores RGBA empacados
GL_UNSIGNED_INT_8_8_8_8_REV	Valores RGBA empacados
GL_UNSIGNED_INT_10_10_10_2	Valores RGBA empacados
GL_UNSIGNED_INT_2_10_10_10_REV	Valores RGBA empacados

2.11.2.1 Empaquetado de Pixeles

Los bitmaps y pixmaps son rara vez empaquetados de manera justa en memoria. En muchas plataformas de hardware, cada fila de datos de un bitmap debe empezar por una dirección en especial por motivos de desempeño [OPEN07]. La mayoría de los compiladores automáticamente colocan las variables y buffers en direcciones de memoria óptimas para esa arquitectura. OpenGL, por defecto, asume alineaciones de 4 bytes, las cuales son apropiadas para muchos sistemas. OpenGL también permite cambiar la manera en que los píxeles son empaquetados con el método glPixelStore*():

```
glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
```

Tabla 30: Parámetros para glPixelStore*() [WRIG04].

Nombre del Parámetro	Tipo	Valor Inicial
GL_PACK_SWAP_BYTES	GLboolean	GL_FALSE
GL_UNPACK_SWAP_BYTES	GLboolean	GL_FALSE
GL_PACK_LSB_FIRST	GLboolean	GL_FALSE
GL_UNPACK_LSB_FIRST	GLboolean	GL_FALSE
GL_PACK_ROW_LENGTH	GLint	0
GL_UNPACK_ROW_LENGTH	GLint	0
GL_PACK_SKIP_ROWS	GLint	0
GL_UNPACK_SKIP_ROWS	GLint	0
GL_PACK_SKIP_PIXELS	GLint	0
GL_UNPACK_SKIP_PIXELS	GLint	0
GL_PACK_ALIGNMENT	GLint	4

GL_UNPACK_ALIGNMENT	GLint	4
GL_PACK_IMAGE_HEIGHT	GLint	0
GL_UNPACK_IMAGE_HEIGHT	GLint	0
GL_PACK_SKIP_IMAGES	GLint	0
GL_UNPACK_SKIP_IMAGES	Glint	0

2.11.2.2 Magnificación, Reducción y Volteado de Imágenes

OpenGL permite jugar con las imágenes, es decir, modificar el tamaño y orientación de estas. El método usado para estos efectos es glPixelZoom(). Una de las aplicaciones de este comando es la simulación de una lupa (lente de magnificación) para agrandar porciones de la pantalla [ASTL04].

Si se envían parámetros negativos, la orientación de la imagen es afectada:

```
glPixelZoom(-1.0f, -1.0f);
```

Si se envían valores menores a 1, la imagen es reducida. Valores positivos magnifican la imagen:

```
glPixelZoom(0.5f, 2.0f);
```

2.12 Mapeado de Texturas

2.12.1 Introducción

Hasta ahora se ha cubierto gran parte de las prestaciones de OpenGL (renderizado de primitivas, modos de vista, iluminación y color, imágenes y mapas de bits). A continuación, se utilizará estas bases para dar el máximo nivel de realismo a una escena a travez del mapeado de texturas (texture mapping).

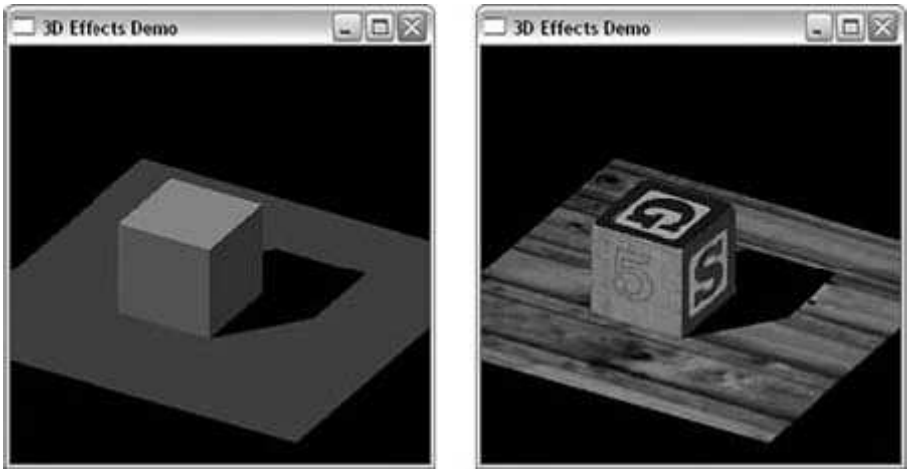


Ilustración 127: Una escena con colores vs una escena con texturas [WRIG04].

Nada brinda tanto realismo a una escena como una textura aplicada a un objeto (véase Artificios 3D: Mapeado de texturas), tal como en la Ilustración 127.

2.12.2 Procedimiento

2.12.2.1 Cargar una Textura

El primer paso hacia la aplicación de textura en una geometría es cargar la textura en memoria. Una vez cargada, esta es vinculada con el objeto u objetos y asignada ciertas propiedades [WRIG04]. Los comandos de OpenGL que permiten cargar una textura desde memoria son glTexImage1D(), glTexImage2D(), glTexImage3D().

```
void glTexImage2D(GLenum target, GLint level, GLint
internalformat,GLsizei width, GLsizei height, GLint border, GLenum
format, GLenum type, void *data);
```

Como se puede apreciar, los parámetros de la función para crear texturas de dos dimensiones (las más comunes), son casi obvios. El parámetro *target* define el tipo de textura (GL_TEXTURE_1D, GL_TEXTURE_2D, o GL_TEXTURE_3D), *level* define el nivel de detalle de la textura, *internalformat* indica el formato de la textura (vease Tabla 31). Las dimensiones *width, height* y *depth* (dependiendo de la función 3D, 2D o 1D) indican el tamaño de la textura. El parámetro *border* permite incrementar las dimensiones para crear un borde, *format* y *type* son similares a los parámetros de glDrawPixels() y sus valores validos se listan en la Tabla 32 y Tabla 33.

Tabla 31: Formatos de textura internos más comunes [WRIG04].

Constante	Significado
GL_ALPHA	Almacena los texels como valores alfa.
GL_LUMINANCE	Almacena los texels como valores de luminancia.
GL_LUMINANCE_ALPHA	Almacena los texels como valores de luminancia y alfa.
GL_RGB	Almacena los texels como valores de componentes RGB.
GL_RGBA	Almacena los texels como valores de componentes RGBA.

Tabla 32: Formatos de texels para glTexImage() [OPEN07]

Constante	Significado
GL_RGB	Los colores estan en orden RGB.
GL_RGBA	Los colores estan en orden RGBA.
GL_BGR /GL_BGR_EXT	Los colores estan en orden BGR.
GL_BGRA /GL_BGRA_EXT	Los colores estan en orden BGRA.
GL_RED	Cada píxel contiene solo el componente rojo.
GL_GREEN	Cada píxel contiene solo el componente verde.
GL_BLUE	Cada píxel contiene solo el componente azul.
GL_ALPHA	Cada píxel contiene solo el componente alfa.
GL_LUMINANCE	Cada píxel contiene solo el componente de luminancia (intensidad).
GL_LUMINANCE_ALPHA	Cada píxel contiene el componente de luminancia seguido del alfa.
GL_STENCIL_INDEX	Cada píxel contiene solo el indice de stencil.

GL_DEPTH_COMPONENT	Cada píxel contiene solo el componente de profundidad.

Tabla 33: Tipos de Datos para los píxeles [OPEN07].

Constante	Significado
GL_UNSIGNED_BYTE	Cada componente de color es un entero positivo de 8 bits.
GL_BYTE	Entero de 8 bits.
GL_BITMAP	Bits unicos, sin datos de color. Igual que glBitmap()
GL_UNSIGNED_SHORT	Entero positivo de 16 bits.
GL_SHORT	Entero de 16 bits.
GL_UNSIGNED_INT	Entero positivo de 32 bits.
GL_INT	Entero de 32 bits.
GL_FLOAT	Flotante de precisión simple.
GL_UNSIGNED_BYTE_3_2_2	Valores RGB empaquetados.
GL_UNSIGNED_BYTE_2_3_3_REV	Valores RGB empaquetados.
GL_UNSIGNED_SHORT_5_6_5	Valores RGB empaquetados.
GL_UNSIGNED_SHORT_5_6_5_REV	Valores RGB empaquetados.
GL_UNSIGNED_SHORT_4_4_4_4	Valores RGBA empaquetados.
GL_UNSIGNED_SHORT_4_4_4_4_REV	Valores RGBA empaquetados.
GL_UNSIGNED_SHORT_5_5_5_1	Valores RGBA empaquetados.
GL_UNSIGNED_SHORT_1_5_5_5_REV	Valores RGBA empaquetados.
GL_UNSIGNED_INT_8_8_8_8	Valores RGBA empaquetados.
GL_UNSIGNED_INT_8_8_8_8_REV	Valores RGBA empaquetados.
GL_UNSIGNED_INT_10_10_10_2	Valores RGBA empaquetados.
GL_UNSIGNED_INT_2_10_10_10_REV	Valores RGBA empaquetados.

Las texturas cargadas no pueden ser utilizadas a menos que se active el estado de texturizado con glEnable() con el tipo de textura a usar: GL_TEXTURE_1D , GL_TEXTURE_2D , o GL_TEXTURE_3D. Solo un tipo puede estar activado al mismo tiempo, sin embargo, OpenGL brinda opciones de multitextura (multitexturing) que se explica más profundamente en [OPEN07].

```
glEnable(GL_TEXTURE_2D);
```

2.12.2.2 Mapeado de Texturas en Geometrías

Al cargar y activar las texturas, pueden ser aplicadas a cualquier primitiva. Sin embargo, es necesario indicar a OpenGL como mapear la textura en las geometrías. Esto se realiza al especificar las coordenadas de textura para cada vértice. Las coordenadas de textura son definidas como *s, t, r,* y *q* (análogo a las coordenadas de vértices *x, y, z,* y *w*) que soportan de 1 a 3 dimensiones y opcionalmente una escala [ASTL04].

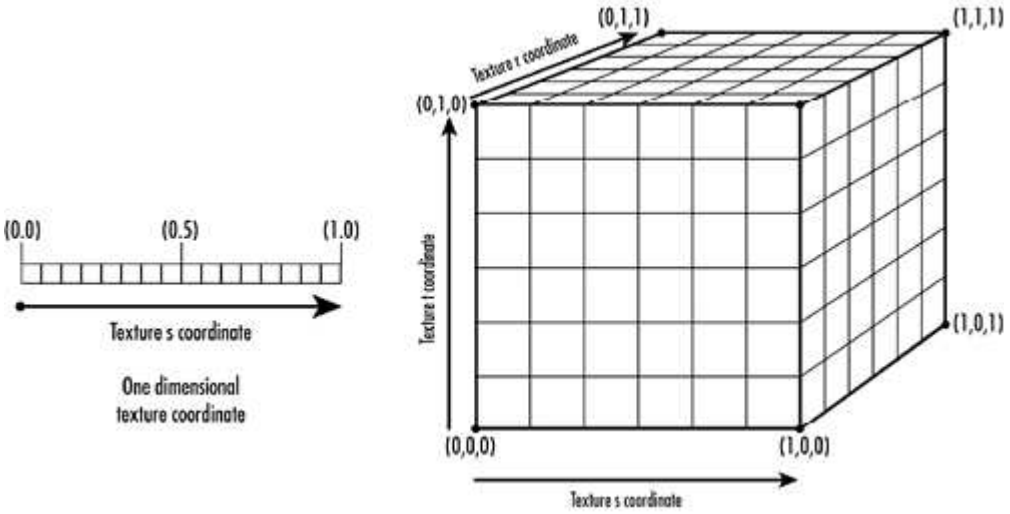


Ilustración 128: Coordenadas de textura [WRIG04].

Para especificar las coordenadas de textura, se utiliza el método glTexCoord*() muy similar a glVertex*():

Ejemplo 22: Coordenadas de textura.

```
glBegin(GL_TRIANGLE_STRIP);
  glTexCoord2f(2.0, 0.0); glVertex3f(2.0, -2.0, -2.0);
  glTexCoord2f(0.0, 0.0); glVertex3f(-2.0, -2.0, -2.0);
  glTexCoord2f(2.0, 2.0); glVertex3f(2.0, -2.0, 2.0);
  glTexCoord2f(0.0, 2.0); glVertex3f(-2.0, -2.0, 2.0);
glEnd();
```

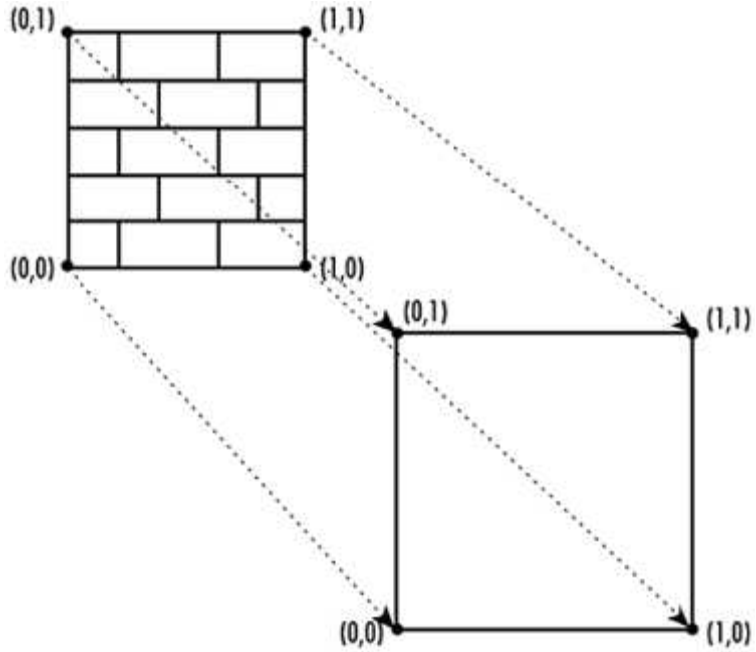


Ilustración 129: Aplicación de coordenadas para rellenar un cuadrado [WRIG04].

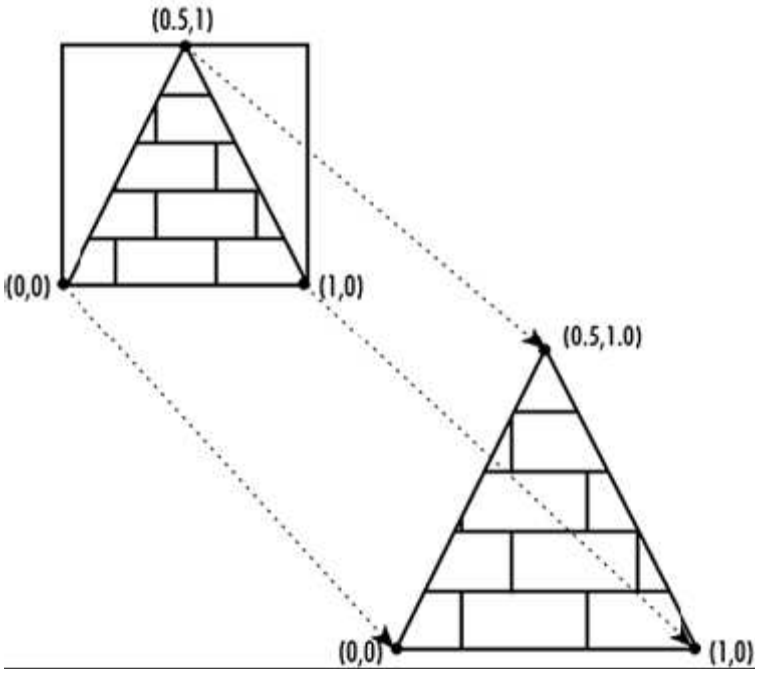


Ilustración 130: Aplicación de coordenadas de manera que se obtenga un triangulo [WRIG04].

2.12.2.3 Matrices de Textura

Las coordenadas de textura también pueden ser trasformadas a través de la matriz de texturas. La pila de matrices de textura funciona de igual manera que con las matrices de proyección o vista de modelos [ASTL04]. De manera que las texturas pueden ser rotadas, trasladadas e inclusive escaladas.

Ejemplo 23: Utilización de la pila de texturas

```
glMatrixMode(GL_TEXTURE);
glPushMatrix();
// Trasladar, rotar o escalar
glPopMatrix();
```

2.12.2.4 Entornos de Textura

OpenGL combina los colores de los texels con los del color de la geometría (material, iluminación, sombreado) [OPEN07]. Para controlar la manera en que esta combinación da lugar, se manipula el modo de entorno de textura con la función glTexEnv*():

Ejemplo 24: Entorno de textura con GL_MODULATE

```
glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

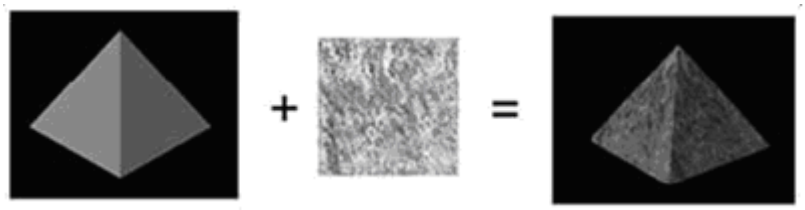


Ilustración 131: Geometría Iluminada + Textura = Textura sombreada [WRIG04]

Uno de las constantes que se puede usar es GL_MODULATE, la cual indica que hay que multiplicar el color del texel por el color de la geometría, claro esta, luego de los cálculos de iluminación. Existen otros modos de entorno de textura

que no se tratarán en particular y son más ampliamente explicados en [ASTL04].

2.12.2.5 Parámetros de Textura

El mapeado de textura involucra mucho más que pegar una imagen en la cara de un polígono. Muchos parámetros afectan las reglas y comportamientos de los mapas de textura cuando son aplicados [ASTL04]. Estos parámetros son asignados a través del método glTexParameter*(). El primer argumento de este comando es el modo de textura al cual se va a asignar ciertas propiedades y puede ser GL_TEXTURE_1D, GL_TEXTURE_2D, o GL_TEXTURE_3D. Los argumentos siguientes serán revisados a continuación.

2.12.2.6 Filtrado

Distintamente de los pixmaps que se dibujan en el colorbuffer, cuando una textura es aplicada a una geometría, casi nunca existe una correspondencia uno a uno entre los texels en el mapa de textura y los pixeles en la pantalla. En este efecto influye mucho la representación final de la geometría que ha sido texturizada y la distancia desde el observador [ASTL04].

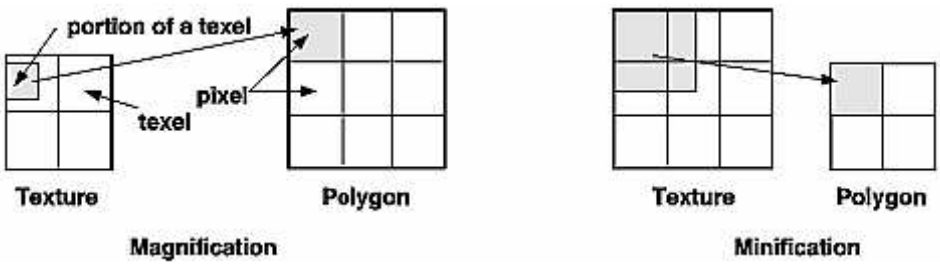


Ilustración 132: Filtrado de texturas [OPEN07]

Entonces, el proceso encargado de calcular el color de los fragmentos de un mapa de textura estirado o magnificado se denomina filtrado de texturas. Usando el método de parametrización de texturas, OpenGL permite asignar filtros de magnificación o minimización GL_TEXTURE_MAG_FILTER y GL_TEXTURE_MIN_FILTER, y sus valores más comunes son GL_NEAREST y GL_LINEAR:

Ejemplo 25: Filtrado de texturas

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
```

Otros valores están disponibles para los filtros, cuya explicación y la forma de cómo internamente calculan la selección de píxeles esta más ampliamente definido en [OPEN07].

2.12.2.7 Modos de Envoltura de Texturas

Generalmente, al asignar las coordenadas de una textura que se va a aplicar a un polígono, se especifican coordenadas entre 0.0 y 1.0. Si las coordenadas salen de este rango, OpenGL las maneja de manera diferente, de acuerdo al modo de envoltura asignado previamente [ASTL04].

Es posible definir un modo de envoltura para cada coordenada individualmente con el método glTexParameteri con los parámetros L_TEXTURE_WRAP_S, GL_TEXTURE_WRAP_T, o GL_TEXTURE_WRAP_R. El modo de envoltura luego puede tener los siguientes valores: GL_REPEAT, GL_CLAMP, GL_CLAMP_TO_EDGE, o GL_CLAMP_TO_BORDER.

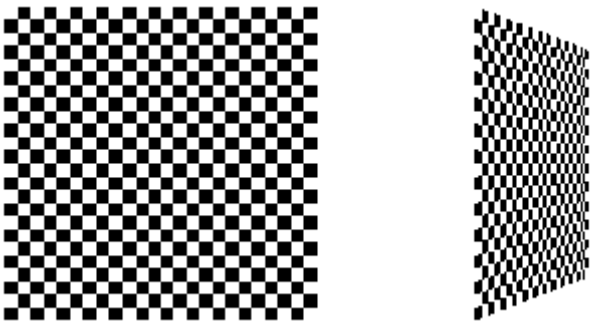


Ilustración 133: Modo de envoltura GL_REPEAT [OPEN07]

Ilustración 134: Modo de envoltura GL_CLAMP [OPEN07]

Ilustración 135: Modo de envoltura GL_REPEAT y GL_CLAMP combinados [OPEN07]

2.12.3 Mipmapping

Las texturas pueden ser vistas, a diferentes distancias en una escena desde el punto de vista. En una escena dinámica, mientras los objetos texturizados se mueven más lejos del punto de vista, el mapa de textura debe reducir su tamaño de manera correspondiente con el tamaño de la imagen proyectada [ASTL04].

Para poder lograr esto, OpenGL debe filtrar el mapa de textura hacia un tamaño adecuado para luego ser aplicado al objeto, sin penalidades en calidad visual. Por ejemplo al renderizar un muro de ladrillo, se utilizaría una textura grande de 128x128 píxeles para cuando el muro se encuentre cerca del punto de vista. Pero si el muro es trasladado lejos del punto de vista, las texturas filtradas pueden cambiar abruptamente de tamaño en ciertos puntos, lo que le resta realismo a la escena.

Para evitar estos inconvenientes, se puede especificar una serie de texturas prefiltradas de resoluciones decrecientes llamadas *mipmaps*. El termino mipmap fue concebido por Lance Williams, cuando introdujo la idea en su trabajo "Pyramidal Parametrics" (SIGGRAPH 1983 Proceedings) [OPEN07]. *Mip* proviene del Latín *multim im parvo*, que significa "muchas cosas en un pequeño lugar". Mipmapping además utiliza métodos muy ingeniosos para empaquetar datos de imagen en memoria.

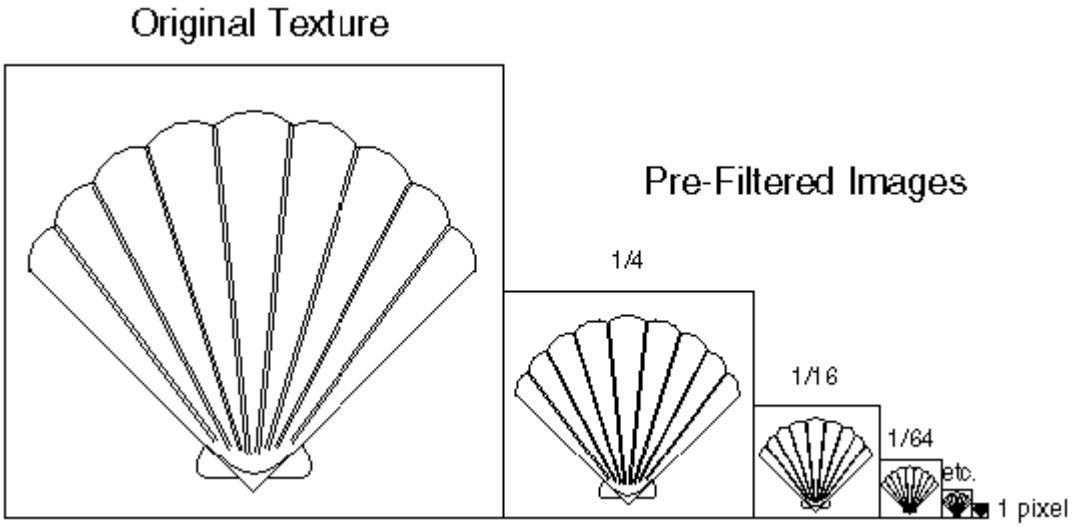


Ilustración 136: Mipmaps [OPEN07]

Para utilizar mipmapping, es necesario proveer todos los tamaños de una textura en potencias de 2 entre su tamaño original hasta el mínimo (1x1 pixel). Por ejemplo, para una textura de 64x64 se debe proveer de mapas de tamaños 32×32, 16×16, 8x8, 4x4, 2x2 y 1x1. No es necesario que las texturas sean cuadradas, pero si deben tener dimensiones que sean potencias de 2, aunque ciertas extensiones permiten texturas de cualquier dimensión [OPEN07].

Ya que la construcción de mipmaps ha sido una operación muy importante (incluida en OpenGL 1.4), OpenGL Utility Library contiene métodos que asisten con esta tarea. Si se ha construido el nivel, o el mapa de más alta resolución, las funciones gluBuild1DMipmaps() y gluBuild2DMipmaps() construirán y definirán la pirámide de mipmaps hasta una resolucion de 1×1 (o 1, para mapas de una sola dimensión). Si la imagen original tiene dimensiones que no

son potencias de 2, gluBuild*DMipmaps() ayuda a escalar la imagen a la potencia de 2 mas cercana [OPEN07].

Ejemplo 26: Especificación de mipmaps de una textura de 64x64.

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 64, 64, 0, GL_RGB,
GL_UNSIGNED_BYTE, textureImage);
```

2.12.4 Objetos de Textura

Hasta ahora se ha revisado como cargar una textura y asignarle los parámetros necesarios. Cargar y mantener el estado de una textura ocupa una considerable porción de recursos en una aplicación, por ejemplo en los videojuegos [ASTL04]. Algunos métodos como, por ejemplo, gluBuildMipmaps() requieren de mucho tiempo de procesamiento y memoria. Además cambiar entre texturas o recargar nuevas inciden en penalidades de desempeño.

Los objetos de textura permiten cargar más de una textura al mismo tiempo y permite cambiar entre ellas muy rápidamente. El estado de la textura es mantenido por el objeto de textura vinculado, el cual es identificado por un entero positivo [OPEN07]. Entonces para asignar un número de objetos de textura se utiliza el método glGenTextures():

```
unsigned int textureArray[3];
glGenTextures(3, textureArray);
```

Con este método basta especificar el número de objetos de textura y un puntero hacia un array de enteros positivos que será llenado con los identificadores de los objetos de texturas.

Para vincular los estados del objeto se utiliza el método glBindTexture():

```
glBindTexture(GL_TEXTURE_2D, textureObject);
```

El primer parámetro puede ser GL_TEXTURE_1D, GL_TEXTURE_2D, o GL_TEXTURE_3D, y el segundo parámetro es el objeto a vincular. De aquí en adelante, cualquier cambio en los parámetros afectara únicamente a la textura actualmente vinculada.

Para eliminar objetos de textura, el método glDeleteTextures() hace el trabajo:

```
glDeleteTextures(3, textureArray);
```

También se puede comprobar los identificadores de los objetos y verificar si son validos con el comando glIsTexture(), el cual retorna verdadero o falso si el parámetro enviado es una textura previamente asignada.

2.12.5 Texturas Residentes

En la mayoría de las implementaciones de OpenGL se soporta una cantidad limitada de memoria de alto rendimiento para las texturas [ASTL04]. Las texturas ubicadas en esta memoria son accesibles muy rápidamente y su rendimiento es muy alto. Ya que esta memoria tiene un espacio limitado, algunas texturas no alcanzarán y se alojaran en memoria del sistema que es más lenta.

Las texturas alojadas en esta memoria de alto desempeño son llamadas residentes. Para determinar si una textura vinculada es residente, basta con llamar a la función glGetTexParameter() con el parámetro GL_TEXTURE_RESIDENT. Comprobar si un grupo de texturas son residentes es más útil que una sola. Para esto es de mucha ayuda el comando glAreTexturesResident() [OPEN07].

2.13 GLUT: OpenGL Utility Library

2.13.1 Introducción

GLUT (OpenGL UtilityToolkit) es un interfaz de programación para escribir programas en OpenGL con independencia del sidstema de ventanas. Esta librería brinda las siguientes características [CHIN98]:

- Ventanas múltiples para el renderizado de OpenGL.
- Procesamientos de eventos mediante callbacks[51].
- Dispositivos de entrada sofisticados.
- Una rutina “idle” y temporizadores.
- Facilidades de menús contextuales.
- Rutinas útiles para varios objetos sólidos o de alambre.
- Soporte para fuentes de bitmap o vectoriales (stroke).
- Funciones misceláneas para el manejo de ventanas, incluyendo overlays (capas superpuestas).

2.13.2 Inicialización y creación de ventanas

Antes de poder abrir una ventana en GLUT, es necesario especificar sus características, buffer único o doble buffer? Modo de color RGBA o indexado? Donde debe estar colocada? Para especificar estas características se utilizan los métodos glutInit(), glutInitDisplayMode(), glutInitWindowSize(), y glutInitWindowPosition() antes de llamar a la función glutCreateWindow() para abrir la ventana [OPEN07].

[51] Un callback es un código ejecutable que es pasado cómo un argumento a otro código. Esto permite al software de la capa de bajo nivel llamar a una subrutina (función) definida en una capa de alto nivel.

Ejemplo 27: Inicialización y creación de la ventana

```
/* Modo RGB, doble buffer y adicionalmente un buffer de profundidad*/
glutInitDisplayMode( GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH );
/* Posición inicial de la ventana en (50,50) */
glutInitWindowPosition( 50, 50 );
/* Tamaño de la ventana (640,480) */
glutInitWindowSize( 640, 480 );
/* Creación de la ventana con un titulo determinado */
glutCreateWindow( "Ventana GLUT" );
```

2.13.3 Manejo de eventos de entrada y de la ventana

Luego de crear la ventana, pero antes de entrar al bucle principal del sistema, es necesario registrar las funciones de callback usando las siguientes rutinas:

- **void glutDisplayFunc(void (*func)(void)):** Especifica la función que se llama cuando el contenido de la ventana necesita ser redibujado debido a ciertos cambios o de manera explicita por glutPostRedisplay().

- **void glutReshapeFunc(void (*func)(int width, int height)):** Especifica la función que es llamada cuando la ventana es redimensionada o movida de manera que se pueda ajustar el viewport (véase Trasformaciones de Vista y de Modelos) y el volumen de visión (véase Proyección en Perspectiva).

- **void glutKeyboardFunc(void (*func)(unsigned int key, int x, int y):** Especifica la función que es llamada cuando una tecla es presionada, Los parámetros x e y indican la posición del ratón cuando se presionó dicha tecla.

- **void glutMouseFunc(void (*func)(int button, int state, int x, int y)):** Especifica la función que es llamada cuando un botón del ratón es presionado o soltado. El parámetro button puede ser GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, o

GLUT_RIGHT_BUTTON. El parámetro state puede ser GLUT_UP o GLUT_DOWN, x e y indican la posición del ratón cuando ocurrió el evento.

- **void glutMotionFunc(void (*func)(int x, int y)):** Especifica la función que es llamada cuando el puntero del ratón se mueve dentro de la ventana y uno o mas botones del mismo son presionados. Este callback invoca a un redibujado de los contenidos de la ventana.

2.13.4 Inicialización y renderizado de objetos tridimensionales

Las siguientes rutinas [CHIN98] provistas por GLUT permiten el renderizado de objetos tridimensionales comunes como cubos o esferas con una sola llamada a función, de manera que no será necesario implementar nuestras propias rutinas para dibujar estos objetos. Estos métodos renderizan todos sus gráficos en modo inmediato (véase Modo Inmediato y Modo Retenido). Cada modelo viene en dos formatos: alambre sin vectores normales de superficie, y sólido con sombreado y normales de superficie. La versión sólida es usada cuando se esta empleando iluminación en la escena. Solo la tetera (teapot) genera coordenadas de textura.

- `void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);`
- `void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);`
- `void glutWireCube(GLdouble size);`
- `void glutSolidCube(GLdouble size);`
- `void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);`
- `void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint nsides, GLint rings);`
- `void glutWireIcosahedron(void);`
- `void glutSolidIcosahedron(void);`
- `void glutWireOctahedron(void);`
- `void glutSolidOctahedron(void);`
- `void glutWireTetrahedron(void);`
- `void glutSolidTetrahedron(void);`

- `void glutWireDodecahedron(GLdouble radius);`
- `void glutSolidDodecahedron(GLdouble radius);`
- `void glutWireCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);`
- `void glutSolidCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks);`
- `void glutWireTeapot(GLdouble size);`
- `void glutSolidTeapot(GLdouble size);`

2.13.5 Manejo de procesos de fondo

Es posible especificar una función para ser ejecutada cuando no hay mas eventos pendientes, por ejemplo cuando el bucle principal de la aplicación esta idle (descansando), con el método glutIdleFunc(). Esto es particularmente util para animaciones continuas u otros procesos de fondo.

2.13.6 Ejecución del Programa

Luego de que todas la configuraciones han sido realizadas, los programas que utilizan GLUT entran en un bucle de procesamiento de eventos, el cual puede ser llamado con el método glutMainLoop().

Ejemplo 28: Cuerpo de un programa GLUT.

```
int main(int argc, char *argv[]){
    glutInitWindowSize(640,480);
    glutInitWindowPosition(40,40);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
    glutCreateWindow("GLUT DEMO");
    glutReshapeFunc(resize);
    glutDisplayFunc(display);
    glutKeyboardFunc(key);
    glutMouseFunc(save_position);
    glutMotionFunc(rotate);
    glutIdleFunc(idle);
    glutMainLoop();
    return 0; }
```

2.14 OpenGL vs Direct3D

A continuación se presenta, de la manera más objetiva y breve posible, las ventajas y desventajas de estas dos APIs 3D, las cuales han se han disputado durante mucho tiempo el liderazgo y popularidad en el mercado (especialmente de videojuegos). La realidad, de cual es la mejor de las APIs 3D es un tema de mucha controversia y discusión que aún no termina y dejo al lector el sacar sus propias conclusiones.

Personalmente escogí esta librería por su portabilidad y desempeño, además de que permite cumplir con los requisitos y objetivos del aplicativo que se ha desarrollado y se describe en el próximo capitulo con más detalle. OpenGL es una librería que brinda muchas características interesantes y técnicas que pueden llegar a ser muy avanzadas pero con resultados impresionantes. Es importante aclarar que el tema de “Gráficos por Computador”, y en especial, los gráficos en tres dimensiones es un tema muy amplio y complejo [WRIG04], es por esto que este trabajo se ha centrado únicamente en OpenGL clásico y especialmente la Iluminación y Color para la simulación de un área de trabajo iluminado.

2.14.1 Ventajas de Direct3D

- Direct3D suporta un número amplio de características.

- Direct3D de la gran control a los programadores sobre la línea de ensamblaje (pipeline) de renderizado, si lo desean. DirectX 9.0 además ofrece shaders para pixels y vértices programables a través de un lenguaje de sombreado de alto nivel parecido al C.

- Microsoft trabaja conjuntamente con los vendedores de hardware grafico para asegurar un nivel más alto de compatibilidad con el hardware grafico mas reciente.

- DirectX 9.0 provee suporte de ejecución de lenguaje común (Common Language Runtime); lo que significa que es programable a través de C# y otros lenguajes .NET.

2.14.2 Desventajas de Direct3D

- Direct3D es un estándar propietario. Microsoft determina su futuro y dirección.

- Direct3D no es portable y seguramente nunca lo será.

- Direct3D es, según varios autores, un API de bajo nivel en comparación de OpenGL, y además es más complicado para los programadores inexpertos.

- Si no se esta familiarizado con COM y la programación de aplicaciones para Windows, Direct3D y en general DirectX le requerirá algún tiempo de costumbre si se desea programar en C/C++.

2.14.3 Ventajas de OpenGL

- OpenGL es un estándar abierto. Un grupo de compañías están a cargo de su especificación y dirección (véase Breve historia de OpenGL).

- OpenGL es altamente portable (véase ¿Qué es OpenGL?).

- OpenGL es un API de alto nivel y fácil de entender y aprender(véase ¿Qué es OpenGL?).

- El uso de OpenGL es muy amplio y variado, desde juegos para computadora, hacia CAD, aplicaciones científicas, visualizaciones y simulaciones, etc. [WRIG04]. DirectX es casi exclusivo para videojuegos para Microsoft Windows.

- OpenGL brinda un buen nivel de emulación por software (si el hardware grafico no esta presente) [WRIG04].

2.14.4 Desventajas de OpenGL

- Al ser un estándar abierto y cuya especificación está a cargo de una grupo de compañías (El Grupo Kronos), su evolución es más lenta.

- OpenGL es extensible, y los vendedores de hardware grafico tiene sus propias extensiones específicas para sus productos [OPEN07].

Tabla 34: Comparación de características entre OPenGL y Direct3D [WEB14]

Característica	OpenGL	Direct3D
Blending de vértices	No	Si
Portable entre Sistemas Operativos	Si	No
Mecanismo de extensiones	Si	Si
Desarrollo	Grupo Kronos	Microsoft
Especificación detallada	Si	No
Iluminación en ambos lados	Si	No
Texturas de volumen	Si	No
Z-buffers independientes del hardware	Si	No
Buffers de acumulación	Si	No
Antialiasing en pantalla completa	Si	Si
Profundidad del campo de visión	Si	Si
Desenfoque de movimiento	Si	Si
Renderizado stereo	Si	No
Atributos de ancho de línea/punto	Si	No
Selección	Si	No
Superficies y curvas paramétricas	Si	No
Cache de geometría	Listas	Buffers de vértices
Emulación del sistema	Si el hardware no esta presente	Deja que la aplicación lo determine
Interfaz	Llamadas a procedimiento	COM
Actualizaciones	Anuales	Anuales
Código fuente	Muestras	Implementación del SDK

Muchas más comparaciones pueden ser encontradas a través del Internet, algunos autores estarán a favor de OpenGL y otros a favor de Direct3D. Tal como se planteo anteriormente, este tema es muy controversial y la selección de un API es especial depende mucho de las necesidades de la aplicación que se le vaya a dar.

Para finalizar quisiera citar a John Carmack[52] quien en una entrevista con Game Informer (www.gameinformer.com) dijo lo siguiente:

"...DX9 es en realidad un muy buen API. Inclusive con la manera de D3D (Direct3D) de hacer las cosas, donde yo se tengo una larga historia de personas que piensan que soy antagonista contra D3D. Microsoft ha hecho un muy buen trabajo de evolución en cada paso –ellos no están preocupados sobre el romper la compatibilidad con versiones anteriores- y es un API bastante limpio. Especialmente me gusta el trabajo que estoy haciendo en el Xbox 360, y es probablemente el mejor API gráfico en su diseño con el cual he trabajado."

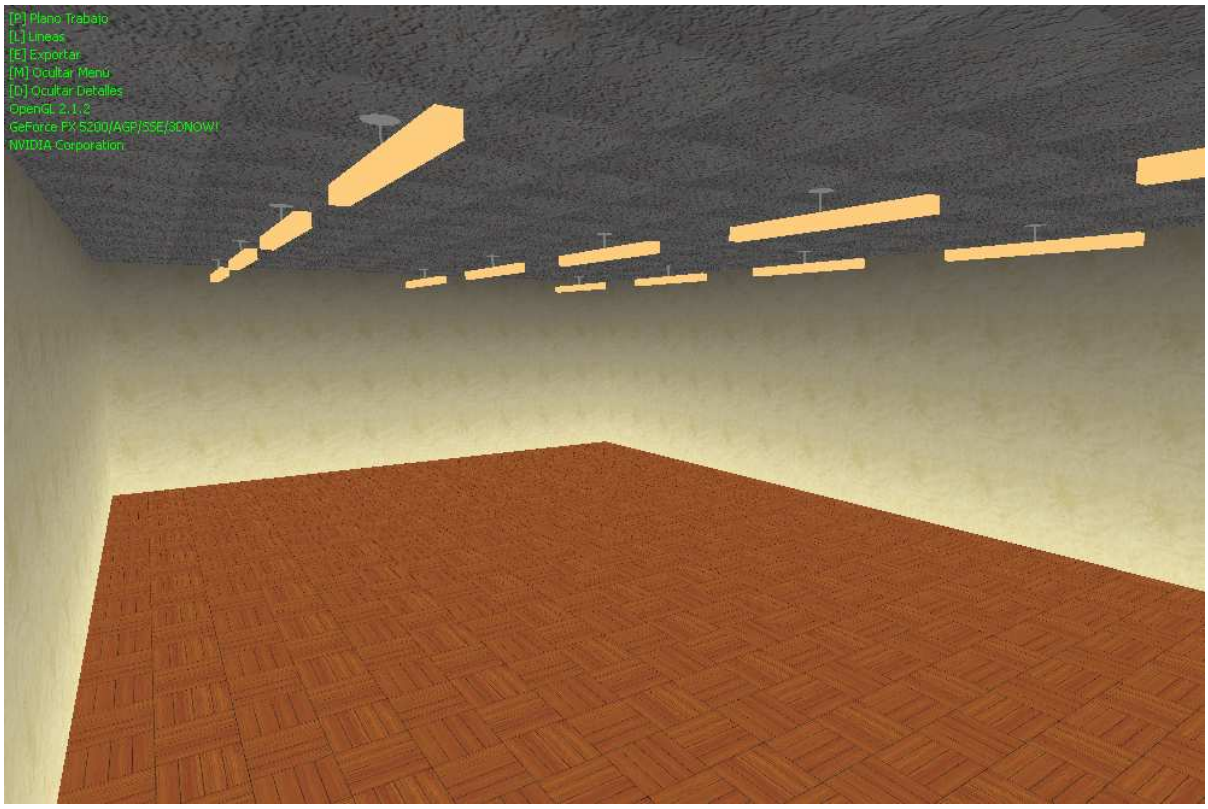
Como una referencia del talento y genialidad de Carmack, vale la pena mencionar que él añadió iluminación dinámica al motor grafico de Quake[53] en solo una hora, según Michael Abrash's “Graphics Programming Black Book”, además fue capaz de portar Quake a OpenGL en solo un fin se semana.

[52] John D. Carmack II (nacido en 1970) es una figura ampliamente reconocida en la industria de los videojuegos. Carmack co-fundó id Software, una empresa de desarrollo de videojuegos, en 1991. Carmack es también muy reconocido por su habilidad como programador, especialmente en el campo de los gráficos tridimensionales.

[53] Quake es un juego de acción en primera persona que fue publicado por id Software el 31 de mayo de 1996. Introdujo algunos de los mayores avances en el género de los juegos en 3D.

CAPITULO III

SIMULACIÓN DE ENTORNOS DE TRABAJO Y DISTRIBUCIÓN EFICIENTE DE LAS FUENTES DE ILUMINACIÓN



3 SIMULACIÓN DE ENTORNOS DE TRABAJO Y DISTRIBUCIÓN EFICIENTE DE LAS FUENTES DE ILUMINACIÓN

3.1 Introducción

El presente capítulo explicará como se desarrolló el aplicativo sobre la simulación de entornos de trabajo y la distribución eficiente de sus fuentes de iluminación, el cual ayudará a hacer cumplir con los niveles de iluminación establecidos (véase Confort Visual) para mejorar la calidad del ambiente laboral de los empleados y precautelar su salud. Además este aplicativo, al realizar una distribución eficiente de las fuentes de iluminación, permitirá mantener costos normales y hasta inferiores en el consumo de energía eléctrica, y por consecuencia, ayudará en la lucha contra el Calentamiento Global.

El aplicativo, cuyo nombre clave es “ZonalCavs”, debido ha que utiliza el Método de las Cavidades Zonales (véase Métodos de Cálculo de Iluminación de Interiores), ha sido desarrollado enteramente con Software Libre, apoyando el Decreto 1014 firmado el 10 de Abril del 2008 por el Presidente Constitucional de la República Ec. Rafael Correa Delgado.

ZonalCavs puede ser compilado y ejecutado en varios sistemas operativos (Microsoft Windows, Linux, Mac OSX, etc.) y es distribuido de manera libre en base a las 4 libertades [WEB17] que brinda el Software Libre.

3.2 Análisis de Requerimientos

3.2.1 Requerimientos del Usuario

Datos de entrada

Geometría del Local:

1. La información sobre la geometría del local que se va a iluminar (largo, alto, ancho, suspensión de las luminarias y altura del plano de trabajo) debe ser ingresada de manera fácil y rápida.

2. Las unidades de medida deben ser consistentes, por ejemplo, en nuestro país la medida oficial de longitud es el metro, y todas las dimensiones ingresadas deben estar definidas en esta unidad. En cuanto a los ángulos, si fuesen requeridos, se definirán en grados.

3. Los ficheros o archivos mantenidos por el sistema deber ser de formato abierto (texto) para una fácil edición o revisión de contenidos.

4. Es necesario considerar que no todos lo locales o ambientes de trabajo son rectangulares, y debe ser posible ingresar datos geométricos de al menos locales en L y circulares.

Reflectancias:

5. El ingreso de datos sobre las propiedades de los materiales de las paredes, techo y piso del local debe ser deductivo, por lo que es indispensable disponer de una base de información sobre distintos materiales y propiedades previamente definidas. Debe ser posible además, seleccionar independientemente un material para cada pared del local y por defecto usar el mismo material para todas las paredes.

Parámetros varios:

6. El sistema deberá desplegar una lista de niveles de iluminación, basados en las normativas vigentes de varios paises (Argentina, Colombia, Ecuador, EEUU, etc), con sus respectivas descripciones, de manera que el usuario pueda seleccionar la opción deseada o ingresar un nivel de iluminación determinado en el caso de usuarios con más conocimientos sobre el tema.

7. De la misma manera debe ser posible seleccionar el factor de mantenimiento (ensuciamiento o degradación) del local o ingresar un valor determinado.

Selección de Luminarias:

8. La selección de la fuente de iluminación a ser utilizada en el proyecto de iluminación interior podrá ser de tres maneras:

    - Seleccionando una luminaria desde el Catálogo del sistema.

    - Selección de la luminaria mediante un archivo fotométrico de formato abierto, de preferencia con el formato europeo EULUMDAT.

    - Seleccionando la luminaria que el sistema recomienda como la más eficaz según la relación entre el flujo luminoso (lumens) y la potencia (watts) que requiere. Esta fuente de iluminación será escogida de entre las luminarias existentes dentro del Catálogo del sistema.

9. El Catálogo de luminarias del sistema debe permitirle al usuario mantener una base de información sobre luminarias varias, con información que se pueda encontrar de manera fácil (flujo luminoso, potencia, temperatura de color, etc.).

Datos de salida

10. Previo el ingreso de toda la información requerida por el sistema (geometría del local, tipo de luminaria, propiedades de los materiales, nivel de iluminación), el sistema calculará la distribución de las luminarias dentro del local de una manera eficiente en términos de iluminación y consumo eléctrico.

11. Es muy importante que el programa presente la distribución de las luminarias de una manera gráfica para poder apreciar el esquema calculado y evaluar su factibilidad.

12. La información detallada sobre el nivel de iluminación alcanzado, número de luminarias necesarias, medidas de distribución y otros datos relevantes a los cálculos realizados por el sistema, deberán ser presentados de manera clara y legible con sus respectivas unidades.

13. Es importante además, que el sistema presente recomendaciones respecto a los otros factores importantes como el deslumbramiento y la ubicación de los puestos de trabajo.

14. Debido a que el sistema se centra en aplicar las funcionalidades de la librería grafica 3D OpenGL, es indispensable una simulación del entorno laboral iluminado en tres dimensiones, la cual deberá permitir al usuario navegar en el espacio tridimensional para poder apreciar cada rincón del local.

3.2.2 Requerimientos del Sistema

1. La información sobre la geometría del local se ingresará mediante controles numéricos para validar el ingreso de dígitos que se encuentren dentro de rangos aceptables por el método de cálculo.

2. Para facilitar la el ingreso de datos, se agruparán los campos de manera lógica, funcional y en el orden de mayor a menor importancia.

3. El programa mostrará valores por defecto para guiar al usuario sobre la manera correcta del ingreso de los datos.

4. Las unidades de medida serán representadas con su respectivo símbolo de manera de un sufijo dentro de cada componente para brindar una mejor visibilidad.

5. Los archivos mantenidos por el sistema tendrán el formato de texto CSV (Valores Separados por Comas), que puede ser editado en cualquier editor de texto o de manera de celdas con software ofimático como OpenOffice Calc o Microsoft Office Excel.

   Los archivos de configuración estarán ubicados en su respectivo directorio denominado **config**.

6. Mediante una lista desplegable el usuario podrá escoger qué material aplicar al techo, suelo o paredes del local. Adicionalmente se proveerá de componentes numéricos para modificar el valor del porcentaje de reflexión del material seleccionado.

7. Por defecto, se utilizará el mismo material para todas las paredes del local, pero, un cuadro de dialogo permitirá la selección de distintos materiales para cada pared. Luego, el cálculo de la reflexión de la

cavidad de pared se realizara mediante las reflectancias medias (véase Método de las Cavidades Zonales).

8. De manera similar al ingreso de datos sobre los materiales y sus reflectancias, se ingresarán los valores del coeficiente de ensuciamiento o mantenimiento del local y el nivel de iluminación que se desea cumplir.

9. Para la selección de la luminaria a utilizar, se presentará al usuario con tres botones etiquetados de manera descriptiva como “Eulumdat” para la importación de archivos fotométricos EULUMDAT, “Catálogo” para abrir el catálogo de luminarias del sistema, y “Sugerencia” para dejar al sistema escoger la fuente de iluminación más eficaz del Catálogo.

10. AL momento de presionar el botón etiquetado como “Eulumdat”, se abrirá un cuadro de dialogo para la selección del archivo fotométrico deseado con el formato EULUMDAT. Luego de cargarse el archivo, se mostrará información respecto a la luminaria seleccionada y los campos sobre flujo luminoso, potencia y temperatura de color, que podrán ser modificados según las necesidades del usuario. además se activará un botón que permitirá visualizar el diagrama de intensidades luminosas (véase Gráficos y Diagramas) de la luminaria, que será de mucha utilidad a usuarios más expertos.

11. Si el botón “Sugerencia” es presionado, se mostrará un mensaje, informando al usuario que se ha realizado una búsqueda de la luminaria más eficaz de entre las provistas en el Catálogo. La selección estará basada en la relación entre flujo luminoso (lumens) y potencia requerida (watts), y la luminaria con el valor más alto será escogida. Quedará a elección del usuario el utilizar esta luminaria o seleccionar otra más acorde a sus necesidades.

12. Cuando el botón “Catálogo” sea presionado, se desplegará un cuadro de dialogo que contiene al Catálogo de luminarias del usuario. Mediante pestañas de brinda al usuario las opciones de “Buscar” luminarias, de acuerdo a varios parámetros de búsqueda (nombre, temperatura de color, flujo luminoso, etc.), y la opción de “Editar” los contenidos del catálogo, permitiéndole al usuario del sistema añadir, editar o eliminar fuentes de iluminación.

13. El Catálogo mantendrá la información más común y que más fácilmente se puede encontrar sobre cualquier tipo de luminaria (nombre, flujo luminoso, temperatura de color, índice de reproducción de color, potencia, dimensiones). Esto impide la obtención del diagrama de intensidades luminosas, debido a la falta de información fotómetrica especializada, por lo que esta opción se encontrará deshabilitada al seleccionar una luminaria del Catálogo.

14. El Catálogo será administrado mediante un motor de base de datos pequeño y que no requiera ningún tipo de configuración o instalación previa por parte del usuario. De esta manera las consultas sobre ciertas luminarias se realizarán de una manera más rápida y confiable que al utilizar archivos secuenciales y otro tipo de almacenamiento mediante archivos.

15. Cuando todos los datos requeridos por el sistema ya hayan sido ingresados, se realizarán los cálculos al momento de presionar el botón “Calcular”. Posteriormente se presentará un cuadro de dialogo que mostrará tres opciones de manera de pestañas (“Distribución”, “Detalles” y “3D”).

16. La pestaña etiquetada como “Distribución” mostrará la distribución de las luminarias de una manera bidimensional con una vista desde la parte superior y mostrando las dimensiones más importantes con sus respectivas unidades.

17. La pestaña “Detalles” mostrará un informe con datos obtenidos por el método de las Cavidades Zonales (véase Método de las Cavidades Zonales), es decir el nivel de iluminación obtenido, el número de luminarias requeridas, la potencia requerida, el nivel de uniformidad. Además se presentará información adicional sobre las reflectancias efectivas de cavidad y las relaciones de máximas luminancias [PUEN01]. En esta pestaña también se mostrarán recomendaciones varias (deslumbramiento, selección de luminarias, ubicación de los puestos de trabajo, etc.) que deben considerarse por el ejecutor del proyecto de iluminación interior.

18. La última pestaña “3D” presentará un simulación en tres dimensiones y en tiempo real (véase 3D en Tiempo Real), sobre el local iluminado utilizando el Modelo de Iluminación de OpenGL (véase El modelo de Iluminación de OpenGL), el cual dará una mejor idea de cómo lucirá el ambiente de trabajo. Dentro de esta pestaña de presentaran opciones para exportar el renderizado actual como una imagen (PNG, JPG, BMP), mostrar el plano de trabajo y renderizar el ambiente usando solamente líneas, como una apreciación poligonal de la escena, muy popular en programas de diseño 3D.

19. El renderizado 3D se realizará de la manera más optimizada posible en términos de desempeño aunque esto influya en la calidad visual de la escena. Con esto se puede garantizar que el sistema sea ejecutado con éxito en ordenadores con pocos recursos de memoria de video o aceleración 3D.

20. El modelo de iluminación de OpenGL será el encargado de iluminar la escena a partir de la información obtenida de la luminaria a utilizar, principalmente su temperatura de color (véase Temperatura de Color).

21. Se proveerá además al usuario la opción de navegar por el entorno 3D mediante el uso del ratón (clic izquierdo + movimiento). También

se podrá realizar acercamiento (zoom) o alejamiento de la escena mediante la rueda del ratón.

22. También es importante mencionar que el usuario podrá modificar ciertos parámetros de OpenGL relacionados con el aplicativo, por ejemplo, la distancia del plano cercano y lejano del volumen de visión (véase Proyecciones de Perspectiva).

23. Dentro del renderizado 3D será necesario restringir el viewport (véase Sistemas de Coordenadas) de manera que la escena no se distorsione al redimensionar la ventana.

24. Adicionalmente, se presentará al usuario la opción de poder guardar en un archivo el proyecto de iluminación en el que está trabajando, para que posteriormente este pueda ser recuperado y pueda continuar o modificar el proyecto según sus necesidades.

3.3 Diseño del sistema

3.3.1 Modelo de Casos de Uso

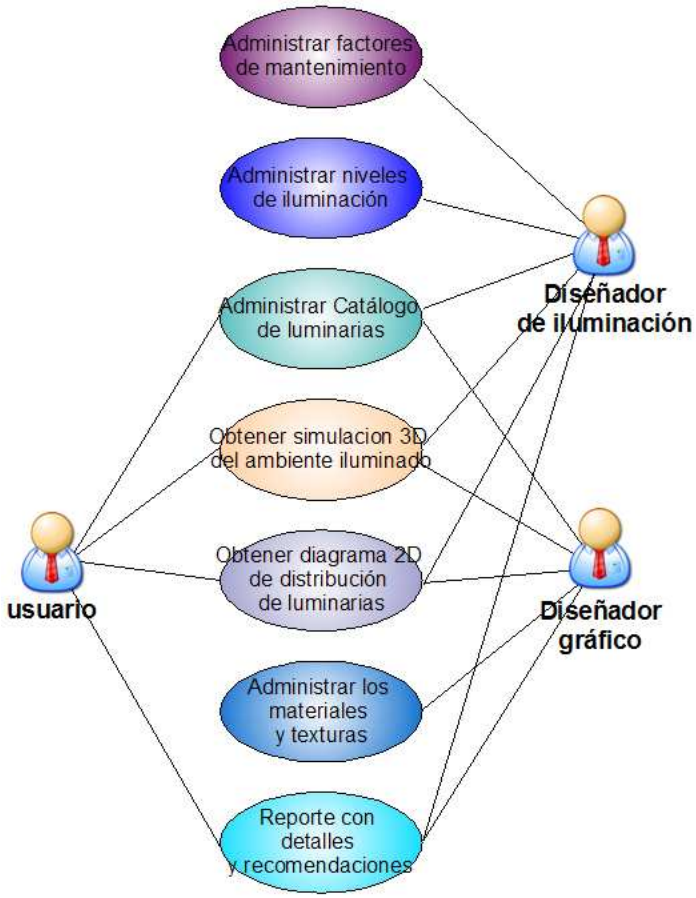


Ilustración 137: Diagrama de Casos de Uso de ZonalCavs.

3.3.1.1 Actores

- **Usuario:** Es cualquier persona que vaya a utilizar el sistema.

- **Diseñador de iluminación:** Es un tipo de usuario especializado y con conocimientos más profundos en el tema de iluminación de interiores y que puede aportar de varias maneras al mejoramiento del sistema.

- **Diseñador grafico:** Es un tipo de usuario con conocimientos sobre diseño grafico y cuya experiencia puede ayudar a mejorar el sistema mediante la creación/edición de propiedades de los materiales (véase Iluminación en OpenGL) y la creación/edición de texturas (véase Mapeado de Texturas).

3.3.1.2 Casos de Uso

- Administración de los factores de mantenimiento/ensuciamiento (descripción y valor) utilizados por el sistema como una guía para el usuario.

- Administración de los niveles de iluminación (descripción y valor) utilizados por el sistema como una guía para el usuario.

- Administración del Catalogo de luminarias (información común y fácil de encontrar) desde el cual el usuario podrá escoger la que mejor se ajusta a sus necesidades de iluminación.

- Obtención de la simulación 3D del ambiente/local iluminado con la distribución resultante de las luminarias.

- Obtención del diagrama de distribución de las luminarias (2D) y sus respectivas dimensiones de emplazamiento.

- Administración de los datos de materiales y texturas utilizadas por el sistema para la simulación y los cálculos respectivos de iluminación.

- Obtención de un reporte con información detallada de los resultados obtenidos y las correspondientes recomendaciones sobre otros factores importantes en el diseño de la iluminación de interiores (véase Confort Visual).

3.3.2 Modelo de Contexto

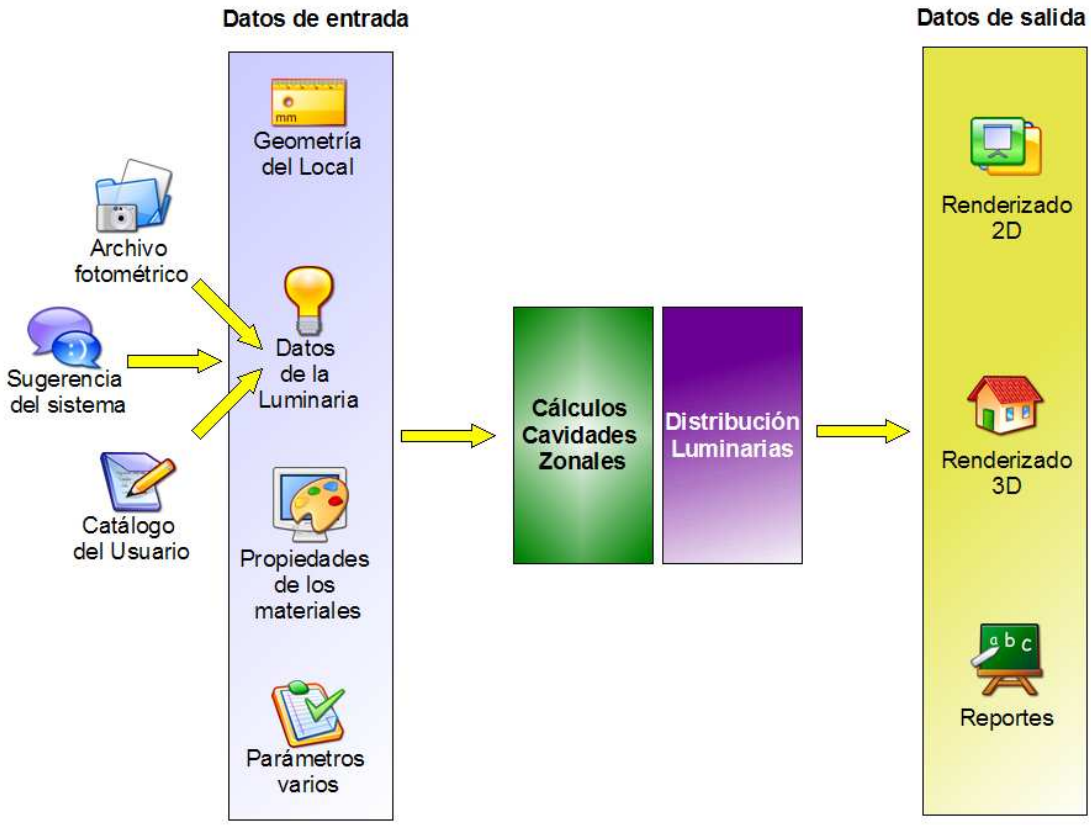


Ilustración 138: Modelo de Contexto de ZonalCavs.

3.3.3 Modelo de Bases de Datos

ZonalCavs utiliza el motor de bases de datos SQLite versión 3 para mantener la información del catalogo de luminarias del usuario, la cual es accedida mediante el controlador QSQLDatabase de Qt y el parámetro QSQLITE.

```
QSqlDatabase db;
db = QSqlDatabase::addDatabase("QSQLITE");
db.setHostName("localhost");
db.setDatabaseName("catalog.db");
db.setUserName("");
db.setPassword("");
bool ok = db.open();
```

La estructura que mantiene la información del catalogo es la siguiente:

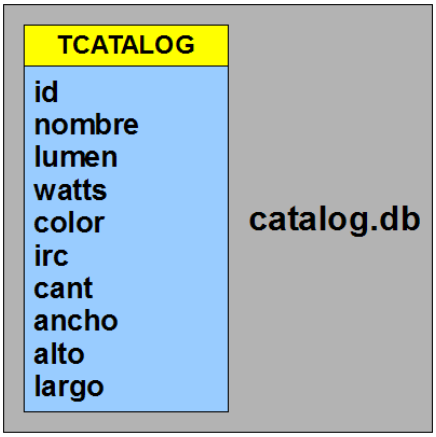


Ilustración 139: Estructura del Catálogo de Luminarias de ZonalCavs.

donde:

- **id:** es el identificador y clave primaria de la tabla.
- **nombre:** es el nombre de la luminaria.
- **lumen:** es el valor del flujo luminoso (lumens) de la luminaria.
- **watts:** es la potencia (watts) de la luminaria.
- **color:** es la temperatura de color (ºK) de la luminaria.
- **irc:** es el índice de reproducción de color IRC de la luminaria.
- **cant:** es el número de luminarias (u) que componen cada set.
- **ancho, alto** y **largo:** son las dimensiones (m) de la luminaria.

La sentencia SQL para la creación del Catálogo es la siguiente:

```
CREATE TABLE tcatalog (
id INTEGER PRIMARY KEY AUTOINCREMENT,
nombre VARCHAR( 255 ) NOT NULL ,
lumen INT UNSIGNED NOT NULL ,
watts INT UNSIGNED NOT NULL ,
color INT UNSIGNED NOT NULL ,
irc   INT UNSIGNED NOT NULL ,
cant TINYINT UNSIGNED NOT NULL ,
ancho FLOAT UNSIGNED NOT NULL ,
alto FLOAT UNSIGNED NOT NULL ,
largo FLOAT UNSIGNED NOT NULL );
```

SQLite nos ofrece un motor de bases de datos pequeño, lo suficientemente veloz para nuestras necesidades y con un nivel alto de portabilidad, ya que la base de datos es contenida en un pequeño archivo, que puede ser compartido entre usuarios sin inconvenientes.

3.3.3.1 Archivos de Datos Adicionales

ZonalCavs requiere una serie de archivos de datos adicionales para su correcto funcionamiento, especialmente debido a las necesidades de información del Método de las Cavidades Zonales. Todos estos archivos adicionales tienen el formato de archivo separado por comas (CSV) por su facilidad de lectura y edición en cualquier software ofimático como OpenOffice Calc o MS Office Excel.

Dentro de los archivos requeridos por el Método de las Cavidades Zonales tenemos:

- **cu_tucuman.csv:** Este archivo mantiene la información sobre los Coeficientes de Utilización (véase Método de las Cavidades Zonales) de un tipo específico de luminaria. Los datos que contiene son producto de mediciones y experimentación en el Laboratorio de Luminotecnia de la Universidad de Tucumán (México).

  Este y los demás archivos requeridos por el Método de las Cavidades Zonales tiene un formato preestablecido y debe ser mantenido si se desea utilizar información más especifica para un tipo de luminaria en especial.

- **rec_tucuman.csv:** Este archivo contiene información sobre las Reflectancias Efectivas de Cavidad (véase Método de las Cavidades Zonales) y también es cortesía del Laboratorio de Luminotecnia de la Universidad de Tucumán.

- **lummed_tucuman.csv:** Este fichero mantiene información sobre las Luminancias Medias (véase Método de las Cavidades Zonales), cortesía del Laboratorio de Luminotecnia de la Universidad de Tucumán.

ZonalCavs además necesita de tres archivos de datos que contienen información que guía al usuario en el ingreso de datos al sistema, y son los siguientes:

- **materiales.csv:** Este archivo mantiene la base de información sobre los materiales que pueden ser utilizados en las paredes, piso y techo del local a iluminar y sus respectivos porcentajes de reflectancia (véase Método de las Cavidades Zonales). Además contiene información requerida por OpenGL, es decir, los valores ambient, diffuse, specular, emission, shininess (véase Iluminación en OpenGL) y el nombre del archivo de textura si fuese necesario.

  La edición de este tipo de archivo es bastante sencilla pero requiere de conocimientos sobre la definición de materiales en OpenGL (véase Iluminación en OpenGL).

- **normativas.csv:** Este fichero es el encargado de mantener un listado de los países, cuyas normativas sobre los niveles de iluminación (véase Método de las Cavidades Zonales) están disponibles para el sistema. Los archivos complementarios a este, como por ejemplo **argentina.csv** contendrán los niveles de iluminación vigentes para el país al que se hace referencia, de esta manera, el usuario no tendrá dificultad al momento de decidir o escoger el nivel de iluminación y la normativa que desea cumplir.

- **factores_man.csv:** Este archivo contiene información sobre los factores de mantenimiento (ensuciamiento) más comunes con su respectiva descripción.

3.3.4 Modelo de Objetos

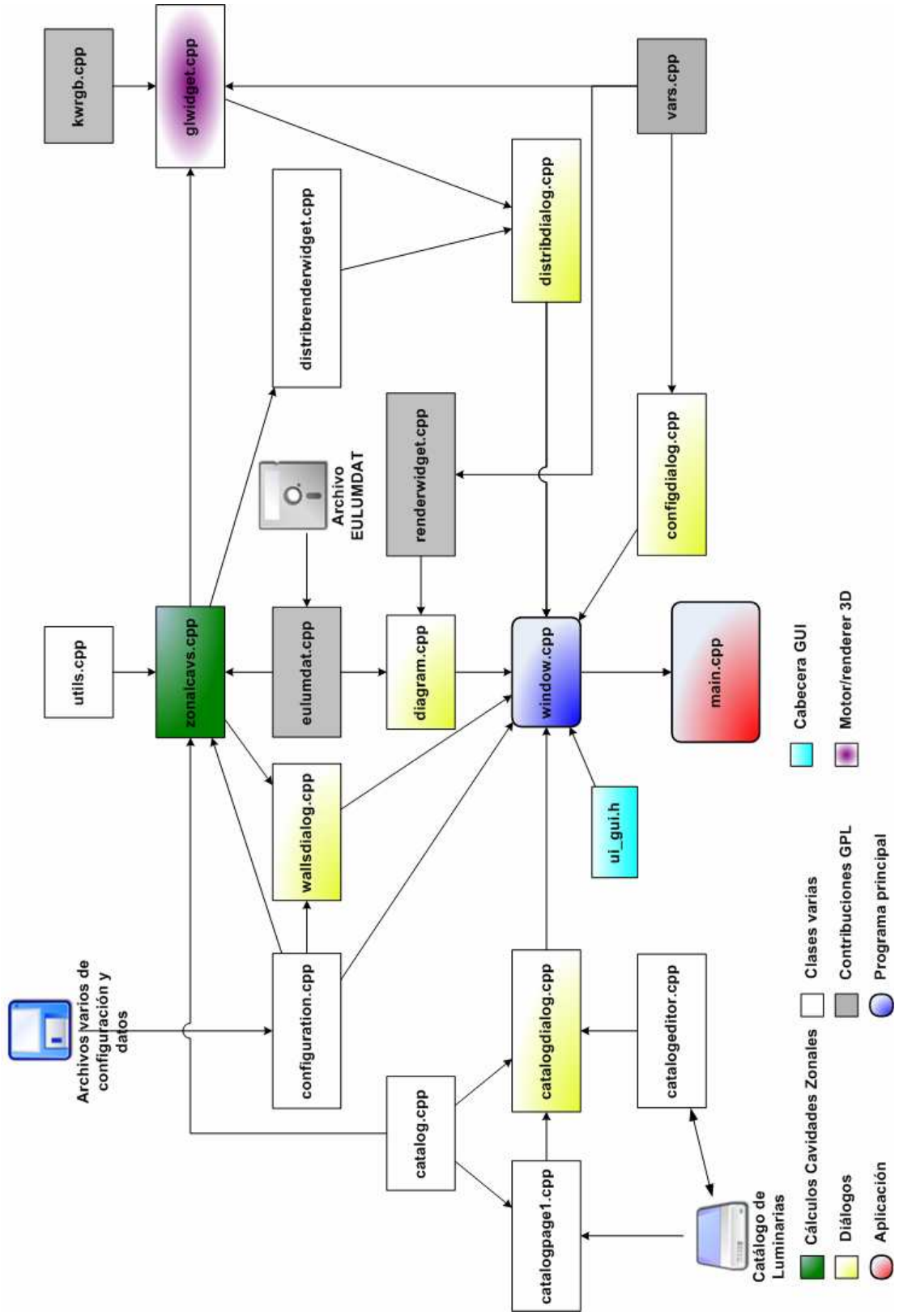


Ilustración 140: Diagrama de Objetos de ZonalCavs.

3.3.4.1 Descripción general de cada clase

CATALOG

Clase para mantener los datos relevantes sobre luminarias en el Catálogo del usuario.

Funciones públicas:

- QString lampReport ();

Atributos públicos

- int id
- QString nombre
- int flujo
- int potencia
- int temperatura
- int irc
- int unidades
- double eficacia
- double ancho
- double alto
- double largo

CATALOGDIALOG

Clase para definir el cuadro de dialogo que contendrá al Catálogo de Luminarias de ZonalCavs.

Funciones públicas:

- CatalogDialog (Catalog &cat, QWidget *parent=0)
- void suggest()

CATALOGEDITOR

Clase que define un componente para la edición del catalogo mediante sentencias SQL. Este componente esta contenido por CatalogDialog.

Slots publicos:

- void addRow ()
- void removeRow ()
- void rowSelected ()
- void updateRow ()

Funciones publicas:

- CatalogEditor (QWidget *parent=0)

CATALOGPAGE1

Clase que define un componente para realizar búsquedas en el catalogo mediante sentencias SQL. Este componente esta contenido por CatalogDialog.

Slots públicos:

- void updateCheckNombre ()
- void updateCheckInt ()
- void updateCheckTemp ()
- void updateCheckPotencia ()
- void updateCheckIRC ()
- void findLamp ()
- bool findBestLamp ()

Funciones publicas:

- CatalogPage1 (Catalog &cat, QDialog *parent=0)
- bool setLamp ()

Atributos públicos:

- QPushButton * cmdBuscar
- QPushButton * cmdAceptar

CONFIG

Clase para manejar los archivos y estructuras de datos sobre niveles de iluminación, materiales y factores de mantenimiento.

Funciones publicas:

- bool loadMaintFactors ()
- bool loadMaterials ()
- bool loadLevels ()
- QList< fm * > getFactors ()
- QList< materials * > getMaterials ()
- QList< levels * > getLevels ()
- QList<normas *> getNormas()

CONFIGDIALOG

Componente que define un cuadro de diálogo para cambiar los parámetros de configuración de OpenGL.

Funciones públicas:

- ConfigDialog (QWidget *parent=0)

DIAGRAM

Clase para definir un cuadro de diálogo que mostrará el Diagrama de Intensidades Luminosas de la luminaria.

Funciones públicas:

- Diagram (Eulumdat &ldt, QWidget *parent=0)

Atributos publicos:

- RenderWidget *render

DISTRIBDIALOG

Clase que define un cuadro de diálogo que contendrá la distribución de luminarias en 2D, 3D y los resultados obtenidos.

Funciones públicas:

- DistribDialog (ZonalCavities *zc, QWidget *parent=0)

DISTRIBRENDERWIDGET

Componente encargado de dibujar la distribución 2D mediante el API de dibujado de Qt. Este componente esta contenido en DistribDialog.

Funciones publicas:

- DistribRenderWidget (ZonalCavities *zc, QWidget *parent=0)

Atributos públicos:

- ZonalCavities *zonal

Funciones protegidas:

- void paintEvent (QPaintEvent *event)

EULUMDAT

Clase que mantiene los datos de la luminaria obtenidos desde un archivo fotométrico EULUMDAT. Cortesía del programa QLumEdit de Krzysztof Struginski.

Funciones públicas:

- QStringList loadFile (QString)
- int saveFile (QString)
- void calcMc ()
- QStringList validate ()
- QString lampReport ()

Atributos públicos:

- QString sIden
- int iItyp, iIsym, iNc, iNg
- double dDc, dDg
- QString sMrn, sLnam, sLnum, sFnam, sDate

- double dL, dB, dH, dLa, dB1, dHC0, dHC90, dHC180, dHC270, dDFF, dLORL, dCFLI, dTILT, *dTLF
- int iN, *iNL
- QString *sTL, *sCA, *sCRG
- double *dWB, dDR [10], *dC, *dG, **dLcd
- int iMc
- QStringList Isimetria, Itipo

GLWIDGET

Clase que define el motor gráfico para la simulación del ambiente laboral iluminado mediante la utilización de OpenGL y el componente QGLWidget que permite el renderizado de OpenGL en cualquier componente de Qt. Este componente esta contenido en DistribDialog.

Slots públicos:

- void setXRotation (int angle)
- void setYRotation (int angle)
- void setZRotation (int angle)
- void setZoom (int z)

Funciones publicas:

- GLWidget (ZonalCavities *zc, QWidget *parent=0)
- QSize minimumSizeHint () const
- QSize sizeHint () const

Atributos publicos:

- ZonalCavities *zonal

Funciones protegidas:

- void initializeGL ()
- void paintGL ()
- void resizeGL (int width, int height)
- void mousePressEvent (QMouseEvent *event)

- void mouseMoveEvent (QMouseEvent *event)
- void wheelEvent (QWheelEvent *event)
- void keyPressEvent (QKeyEvent *event)

KWRGB

Clase encargada de la conversión desde temperatura de color en º Kelvin hacia valores R,G,B. Cortesía de Michael Edwards y traducido a lenguaje C++ por el autor de este trabajo.

Funciones publicas:

- void convertKRGB (int kelvinRef, int kelvinCon, int colorRGB[3])

MAINWINDOW

Clase que define la ventana principal de la aplicación y las instancias de los objetos contenidos en la misma.

Funciones publicas:

- MainWindow (QWidget *parent=0)

Atributos públicos:

- Config * configuration
- ZonalCavities *zonalcavs
- Diagram *diagram
- DistribDialog *distribDialog
- CatalogDialog *catalogDialog
- WallsDialog *wallsdialog
- ConfigDialog *glconfig

RENDERWIDGET

Componente que permite el dibujado del Diagrama de Intensidades Luminosas de una luminaria. Cortesía del programa QLumEdit de Krzysztof Struginski.

Funciones publicas:

- RenderWidget (Eulumdat &ldt, QWidget *parent=0)
- void setVectors ()
- void saveImage (QString, int)

Atributos públicos:

- QVector< QPointF >vectorC0C180
- QVector< QPointF >vectorC90C270
- double max
- Eulumdat * pldt

Funciones protegidas:

- void paintEvent (QPaintEvent *event)

SINGLETON

Clase de propósito general que mantiene parámetros varios de configuración. Cortesía del programa QLumEdit de Krzysztof Struginski.

Atributos públicos:

- int line, lineFNam, lineal, lineCA, lineCRG
- bool fill, legend
- double glNear, glFar
- int glKelvin, glFactor

Funciones amigas:

- Singleton &Vars ()

WALLSDIALOG

Clase que define un cuadro de diálogo para la selección de materiales independientes para cada pared del local y el cálculo de las reflectancias medias.

Funciones publicas:

- WallsDialog (QList< materials * > mats, QWidget *parent=0)
- void reset ()

Atributos públicos:

- double P
- int w1, w2, w3, w4
- double ancho, alto, largo
- QList< materials * > mats

ZONALCAVITIES

Clase encargada de mantener los datos y realizar los cálculos del Método de las Cavidades Zonales a partir de información de luminarias desde el catálogo o desde un archivo fotométrico.

Funciones publicas:

- bool calculate ()
- bool loadECReflectances ()
- bool loadCU ()
- bool loadMedLum ()
- int findECReflectance (float p2, float p1, float ratio)
- float findCU (int pcc, int pw, float k)
- void findLC (int pcc, int pw, float k)
- void findMedLum (int flux, int angle)
- void createReport ()
- void setLampData ()
- void init ()

Atributos públicos:

- DataSource data_source
- Eulumdat *ldt
- Catalog *cat
- QList< materials * > zc_mats

- QString report
- QString lamp_report
- QString error
- float a, c, d
- int alfa
- float h, h1, h2, h3, p1, p2, p3
- int ceil_index, floor_index, w1_index, w2_index, w3_index, w4_index, pcc, pfc, pw
- float k1, k2, k3, fm, fc, Es, E, Eli, CU, CU_final, q1, q2E, po, pf, Lf, Lo, L1i, L2Ei, Lp, Lcr
- int gamma
- float Lpt, Lpl, PT
- int R1, R2, R3, lamp_lumen, numLamps, lamp_temp, N
- QString lamp_name
- float lamp_power, lamp_w, lamp_h, lamp_l, lamp_efic, lamp_irc, spacing
- int NumLa, NumLL
- float sep_a, sep_L, sep_a_w, sep_L_w

3.3.5 Modelo de Interfaces Gráficas de Usuario

3.3.5.1 Flujo de Ventanas

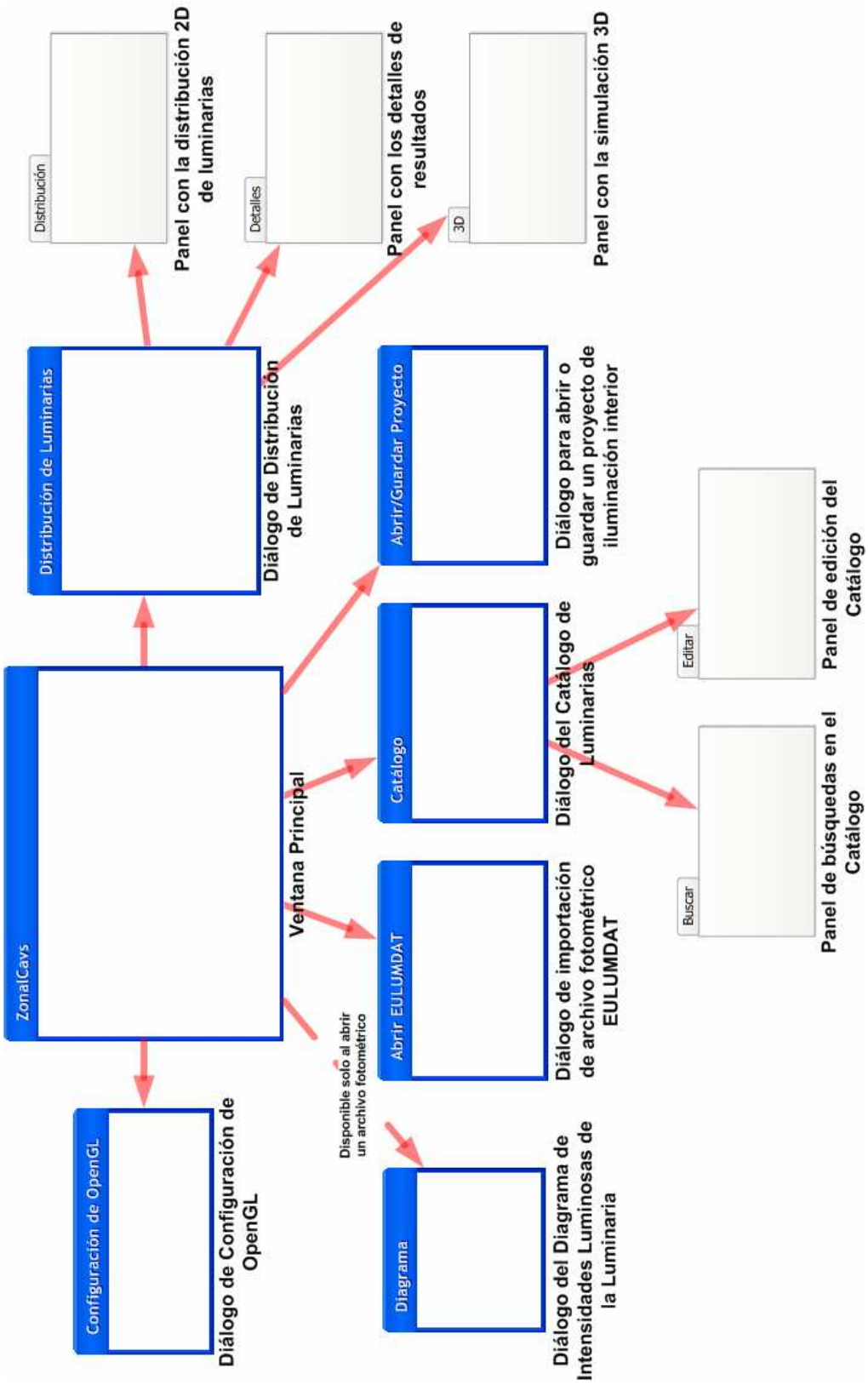


Ilustración 141: Flujo de ventanas de ZonalCavs

3.3.5.2 Ventanas y cuadros de diálogo

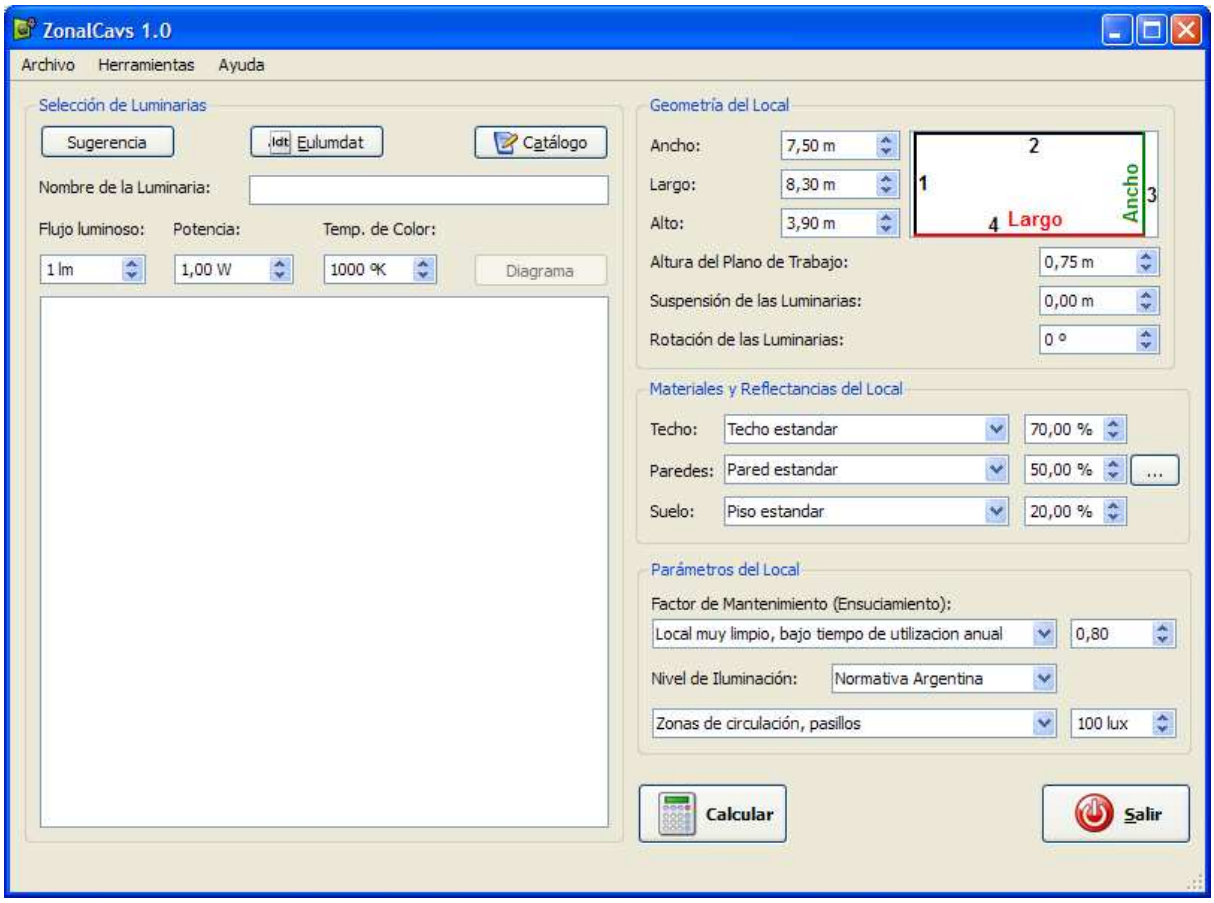


Ilustración 142: Ventana principal

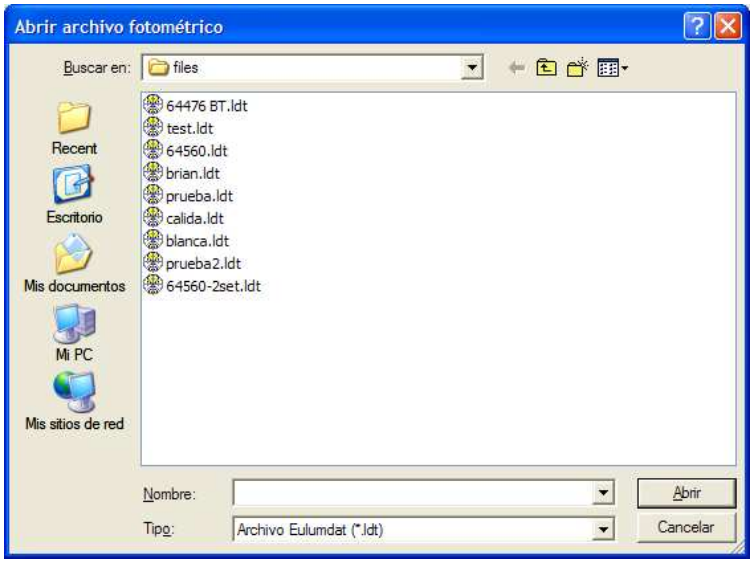


Ilustración 143: Cuadro de diálogo para abrir un archivo fotométrico EULUMDAT

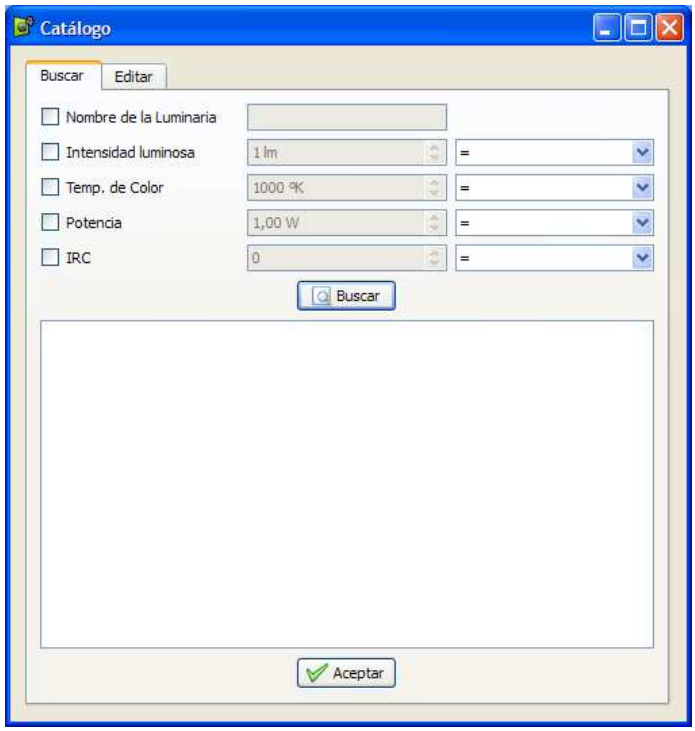


Ilustración 144: Cuadro de diálogo del Catálogo, panel de búsqueda de luminarias.

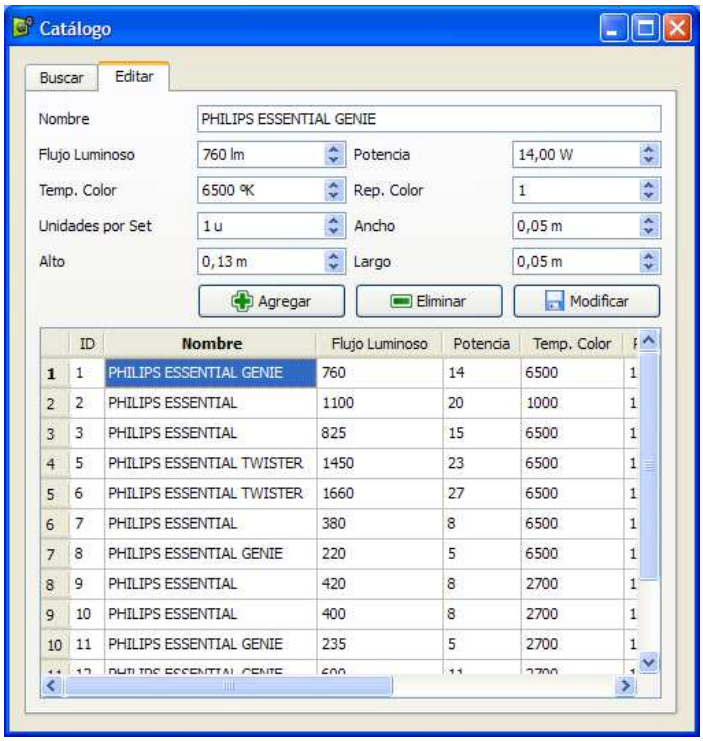


Ilustración 145: Cuadro de diálogo del Catálogo, panel de edición del Catálogo.

Ilustración 146: Cuadro de diálogo de configuración de parámetros de OpenGL.

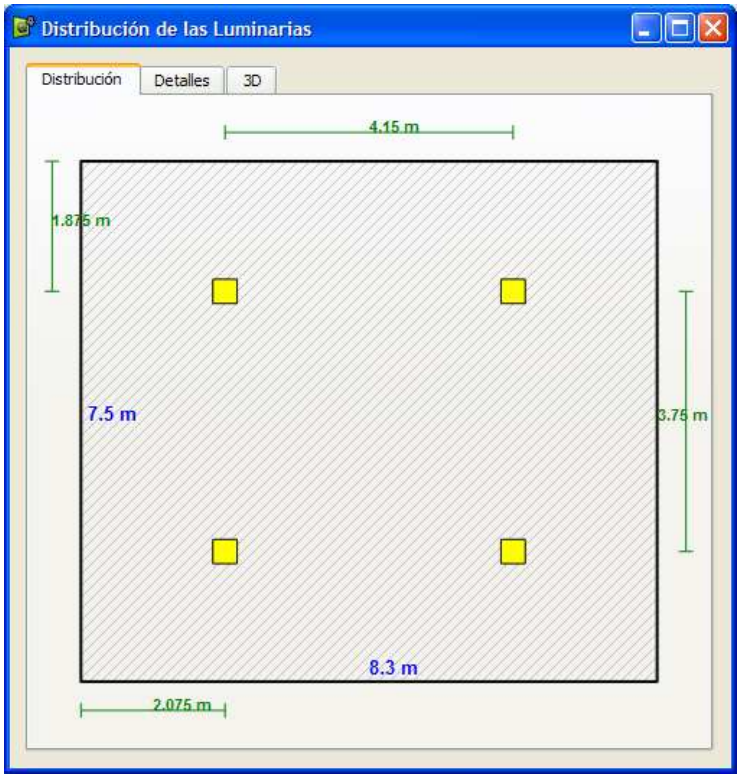
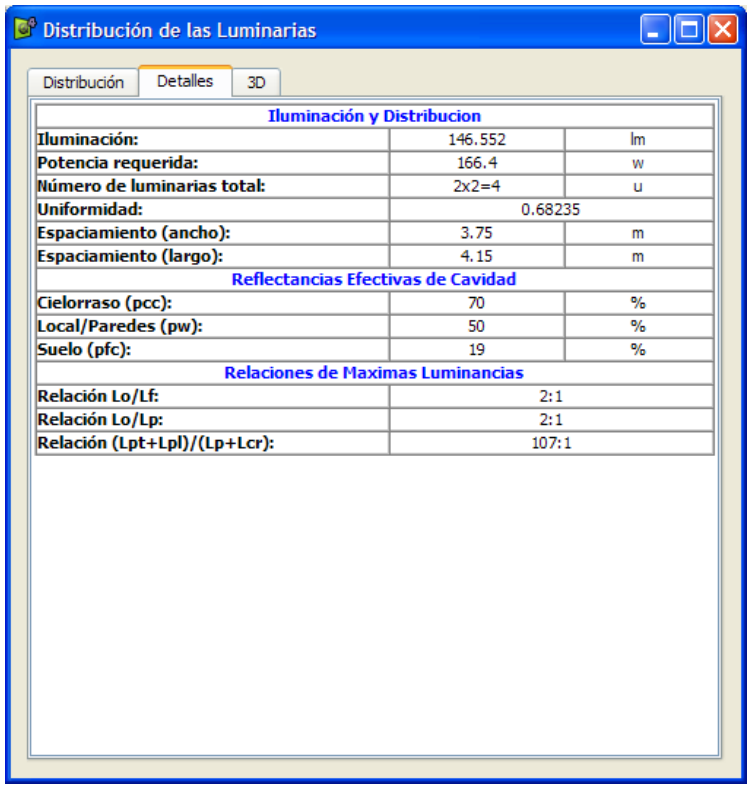


Ilustración 147: Cuadro de diálogo de Distribución de Luminarias, panel de distribución 2D.



Iluminación y Distribucion		
Iluminación:	146.552	lm
Potencia requerida:	166.4	w
Número de luminarias total:	2x2=4	u
Uniformidad:	0.68235	
Espaciamiento (ancho):	3.75	m
Espaciamiento (largo):	4.15	m
Reflectancias Efectivas de Cavidad		
Cielorraso (pcc):	70	%
Local/Paredes (pw):	50	%
Suelo (pfc):	19	%
Relaciones de Maximas Luminancias		
Relación Lo/Lf:	2:1	
Relación Lo/Lp:	2:1	
Relación (Lpt+Lpl)/(Lp+Lcr):	107:1	

Ilustración 148: Cuadro de diálogo de Distribución de Luminarias, panel de detalles de los resultados.

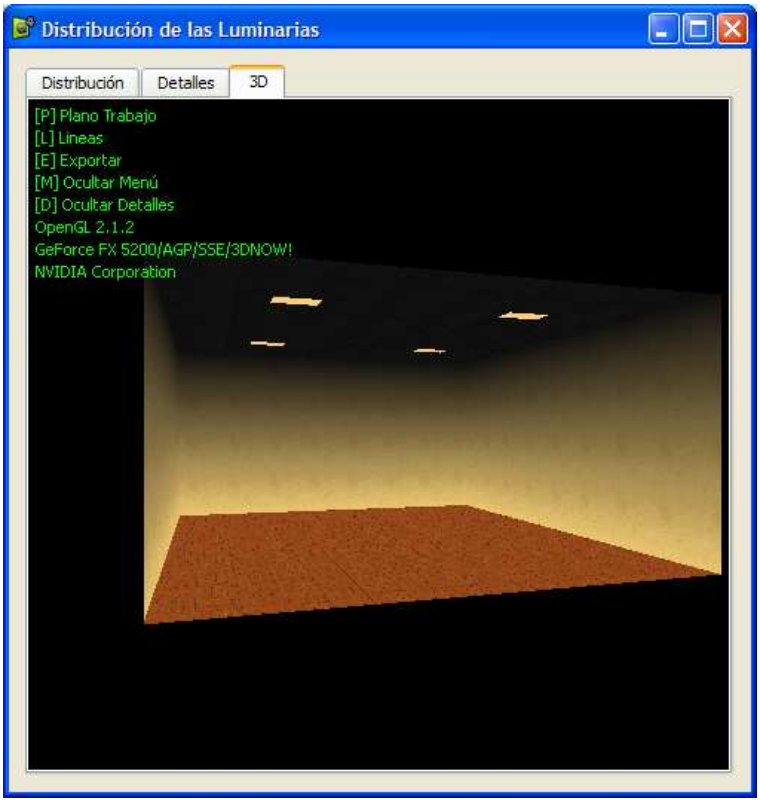


Ilustración 149: Cuadro de diálogo de Distribución de Luminarias, panel de distribución 3D.

3.3.6 Arquitectura del Sistema

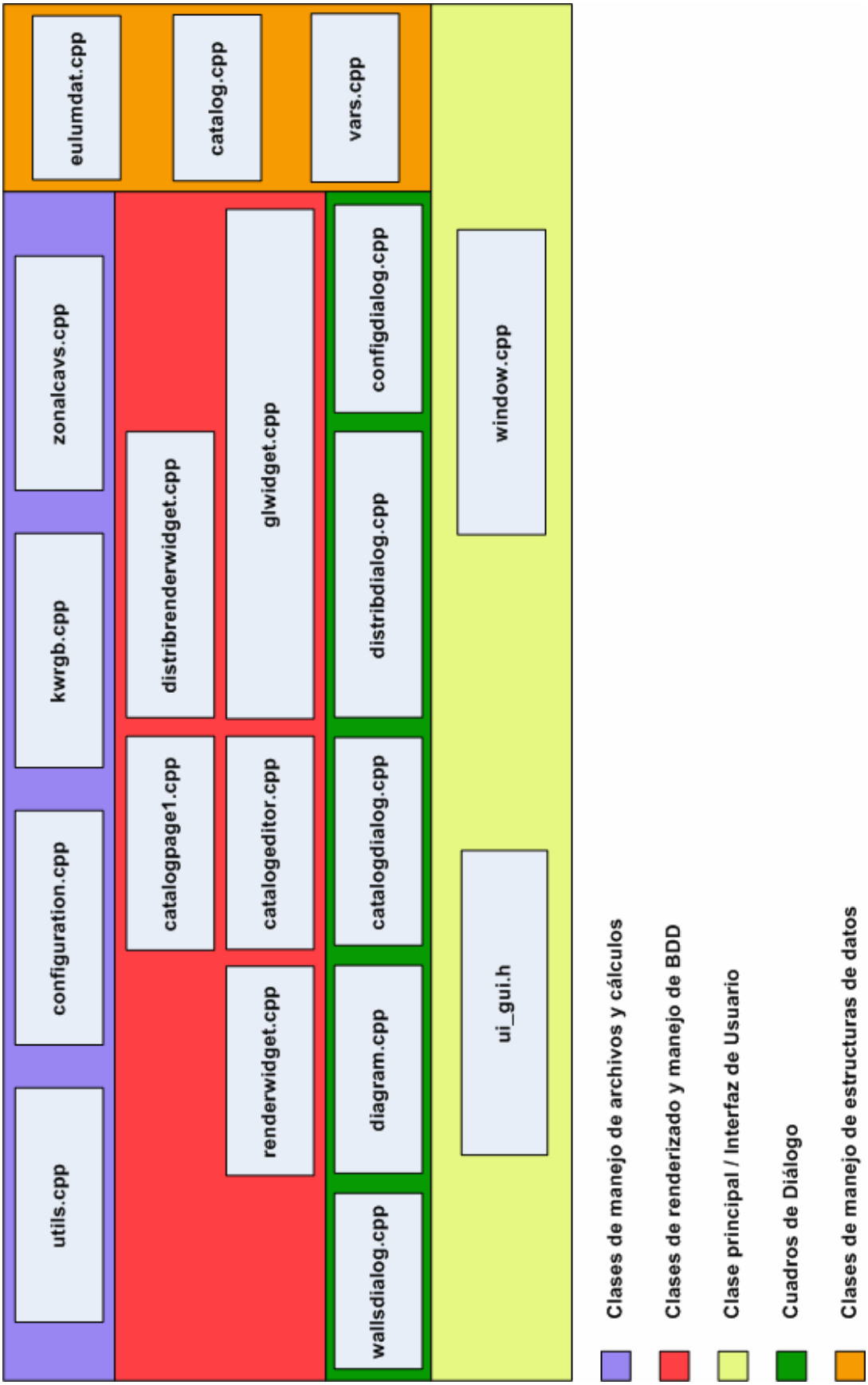


Ilustración 150: Arquitectura general de ZonalCavs

3.4 Implementación

3.4.1 Herramientas y Librerías de Desarrollo

Para la implementación de ZonalCavs fue necesario la utilización de las siguientes herramientas y librerías de desarrollo:

- Qt Open Source Edition versión 4.3.4
- Mingw32 3.8
- Dev C++ 4.9.9.2
- GNU gcc
- The Gimp 2.4.5
- SQLite 3
- OpenGL 2.0

3.4.1.1 Qt Open Source Edition

Qt es una biblioteca multiplataforma para desarrollar interfaces gráficas de usuario. Fue creada por la compañía noruega Trolltech. Qt es utilizada en KDE, un entorno de escritorio para sistemas como GNU/Linux o FreeBSD, entre otros. Utiliza el lenguaje de programación C++ de forma nativa y además existen bindings para C, Python (PyQt), Java (Qt Jambi), Perl (PerlQt), Gambas (gb.qt), Ruby (QtRuby), PHP (PHP-Qt) y Mono (Qyoto) entre otros.

El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML y una multitud de otros para el manejo de ficheros, además de estructuras de datos tradicionales [WEB01].

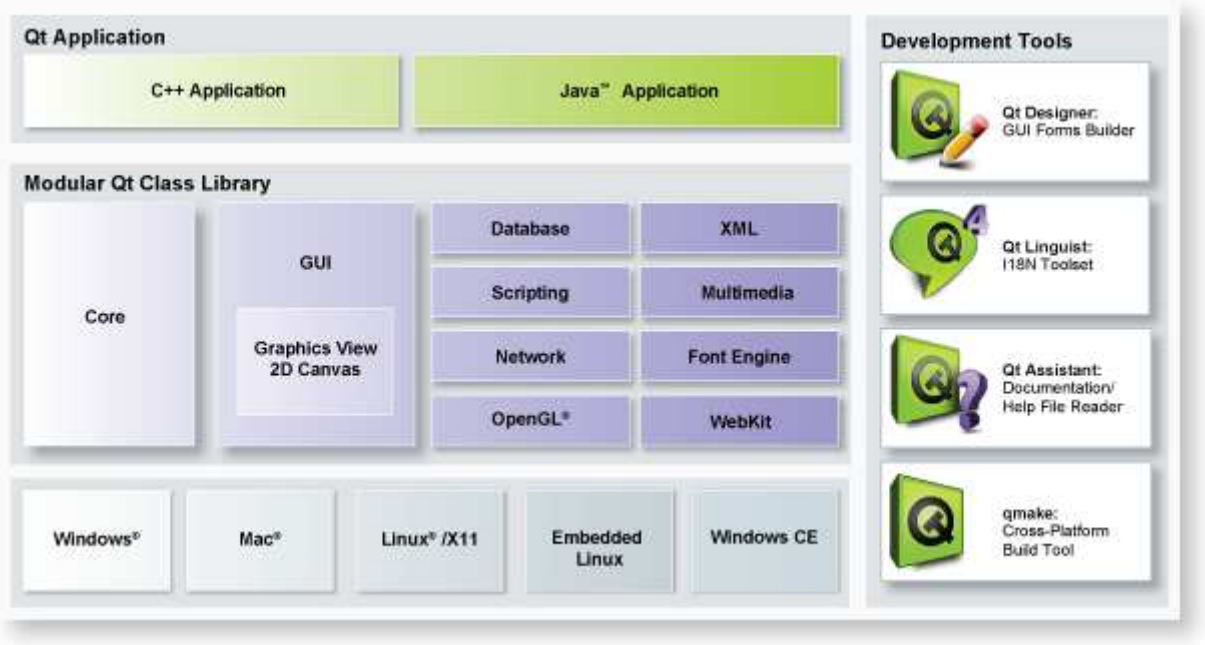


Ilustración 151: Arquitectura de Qt [WEB18]

Qt fue escogida como la librería para el desarrollo de ZonalCavs debido a su portabilidad, facilidad de uso, disponibilidad de herramientas como Qt Designer para el diseño de interfaces de usuario, Qt Assistant que incluye la documentación y ayuda de una manera bien organizada y accesible. Otro punto a favor es la forma nativa en que trabajan las aplicaciones creadas con este framework, lo que le da “Look and Feel” nativo en Windows, Linux y Mac. Además, Qt ofrece la capacidad de crear aplicaciones que incluyan aceleración grafica a través de OpenGL, de manera que OpenGL se encargará del renderizado 3D y Qt se encargará de las tareas de gestión de ventanas.

La ultima versión disponible de Qt puede ser descargada del sitio web de Trolltech http://trolltech.com/downloads.

3.4.1.2 Mingw32

MinGW o MinGW32 (Minimalist GNU for Windows) es la implementación de los compiladores GCC para la plataforma Win32, que permite migrar la capacidad de este compilador en entornos Windows. Es un fork de Cygwin en su versión 1.3.3. Además MinGW incluye un conjunto de la API de Win32, permitiendo un

desarrollo de aplicaciones nativas para esa plataforma, pudiendo generar ejecutables y librerías usando la API de Windows [WEB01].

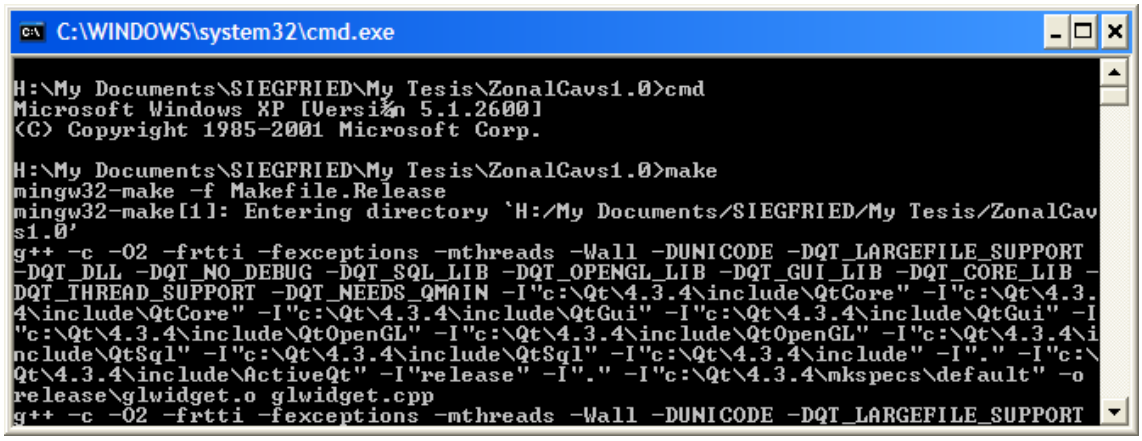


Ilustración 152: Compilación de ZonalCavs mediante el comando MAKE.

Este compilador fue escogido principalmente por ser un compilador libre y gratuito. La ultima versión disponible puede ser descargada desde su sitio web http://www.mingw.org/download.shtml.

3.4.1.3 Dev C++

Bloodshed Dev-C++ es un entorno de desarrollo integrado (IDE) para programar en lenguaje C/C++. Usa MinGW que es una versión de GCC (GNU Compiler Colletion) como su compilador. Dev-C++ puede además ser usado en combinación con Cygwin y cualquier compilador basado en GCC.

El Entorno está desarrollado en el lenguaje Delphi de Borland. Tiene una página de paquetes opcionales para instalar, con diferentes bibliotecas de código abierto [WEB01].

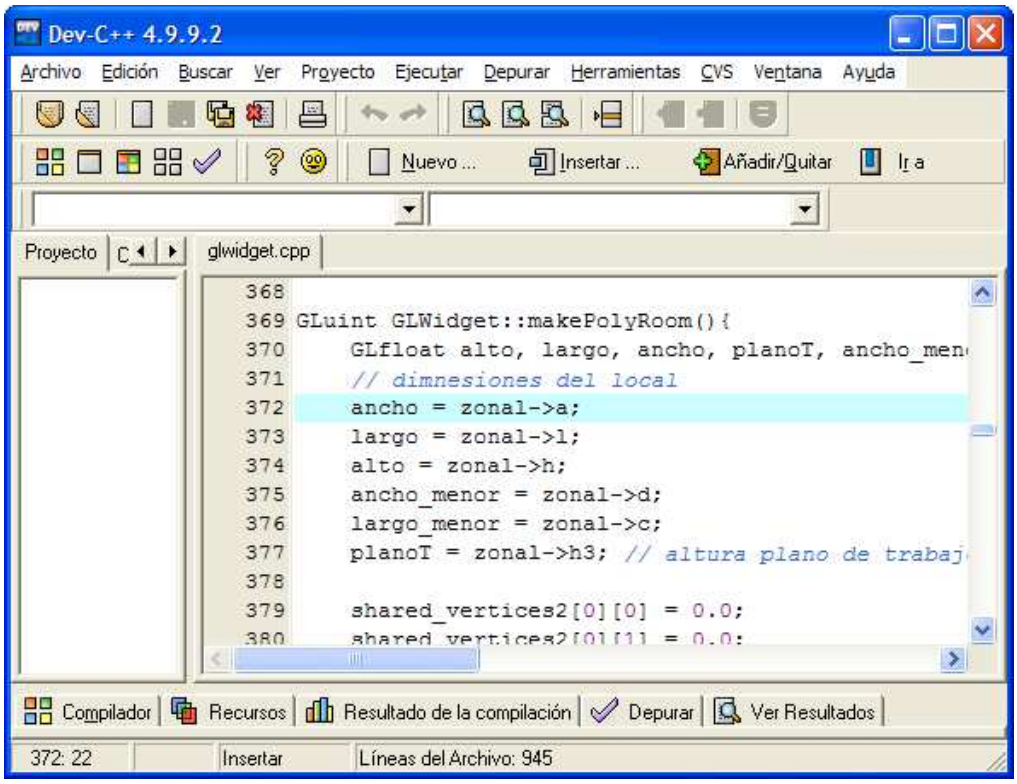


Ilustración 153: Edición del código fuente de ZonalCavs en Dev C++.

Este IDE fue escogido como un editor más que como un IDE aunque sus capacidades de entorno son excelentes y trabaja muy bien con distintos compiladores libres y comerciales.

Dev C++ puede ser descargado sin costo alguno desde su sitio web http://www.bloodshed.net/download.html.

3.4.1.4 The Gimp

GIMP (GNU Image Manipulation Program) es un programa de edición de imágenes, tanto dibujos como fotografías. Es un programa libre y gratuito, englobado en el proyecto GNU y disponible bajo la Licencia pública general de GNU.

La primera versión de GIMP se desarrolló para sistemas Unix y fue pensada especialmente para GNU/Linux. Existen versiones totalmente funcionales para Windows, para Mac OS X y para otros sistemas operativos, haciéndolo el programa de manipulación de gráficos disponible en más sistemas operativos.

Se le puede considerar como la alternativa más firme para Photoshop, aunque posee una interfaz muy diferente.

La biblioteca de controles gráficos GTK, desarrollada para GIMP, dio origen al entorno de escritorio de GNOME [WEB01].

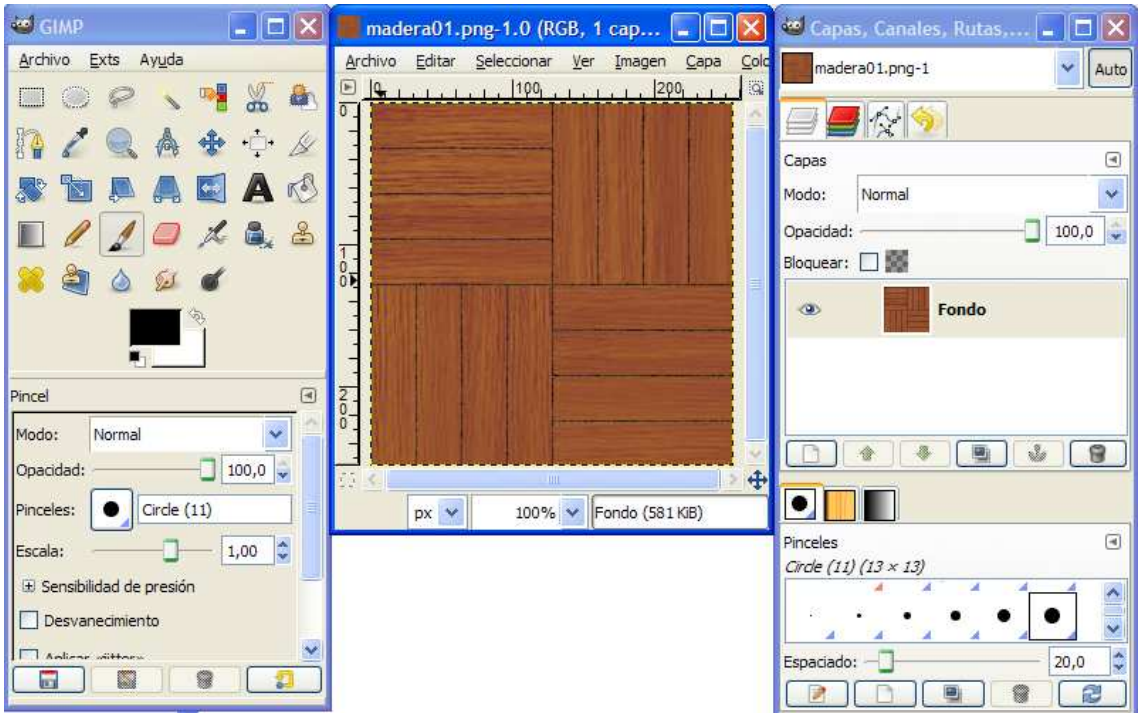


Ilustración 154: The Gimp en la edicion de texturas de ZonalCavs.

Este programa de edición de imágenes fue escogido por su portabilidad, facilidad de uso y por ser libre. Fue utilizado para la edición de las texturas utilizadas en ZonalCavs, por ejemplo al recortar el tamaño de texturas de dimensiones mayores de 256x256 pixels o maximizar texturas de tamaño inferior. También fue muy útil para hacer que las texturas encajen mejor entre si, de manera que no se pueda diferenciar con facilidad donde empieza o donde termina una textura.

Gimp es gratuito y puede ser descargado desde su sitio web http://www.gimp.org/downloads/.

3.4.1.5 SQLite

SQLite es un sistema de gestión de bases de datos relacional compatible con ACID, y que está contenida en una relativamente pequeña (~500kb) librería en C. SQLite es un proyecto de dominio público creado por D. Richard Hipp.

A diferencia de los sistemas de gestión de base de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la librería SQLite se enlaza con el programa pasando a ser parte integral del mismo. El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

En su versión 3, SQLite permite bases de datos de hasta 2 Terabytes de tamaño, y también permite la inclusión de campos tipo BLOB.

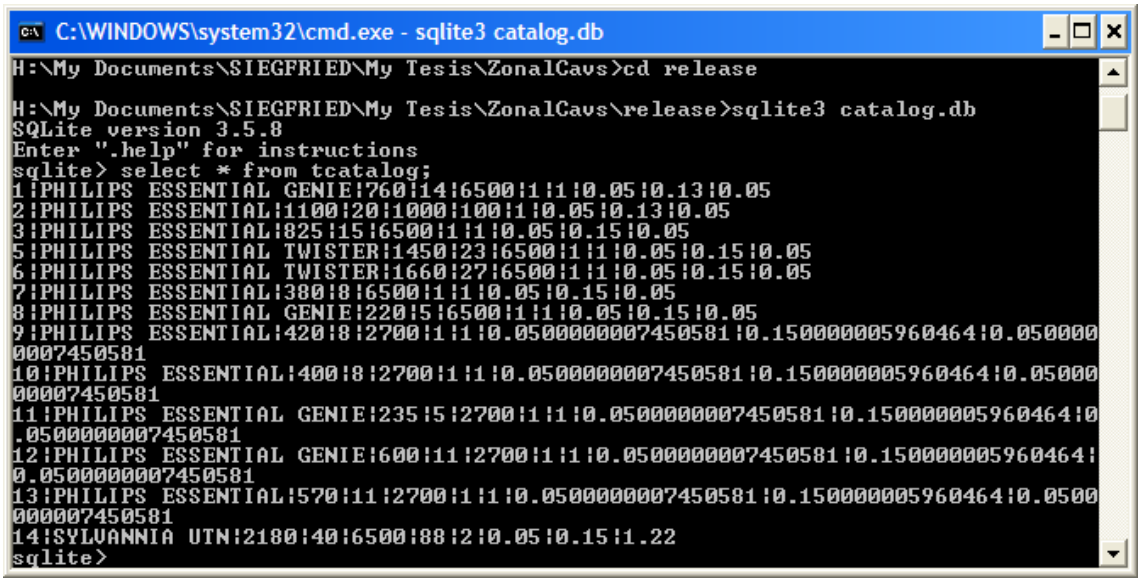


Ilustración 155: Consulta SQL dentro del Catálogo de Luminarias (SQLite3).

SQLite al cumplir con el requisito de portabilidad del catálogo fue una excelente elección que además le dio a ZonalCavs la capacidad de ejecutar sentencias SQL especialmente para la realización de búsquedas según ciertos parámetros. Su velocidad y desempeño son más que suficientes para las necesidades de este aplicativo. Es de notar que SQLite trabaja muy bien conjuntamente con Qt, ya que este tiene ya incorporado el acceso a este gestor de base de datos.

SQLite es además libre y gratuito y se lo puede descargar desde http://www.sqlite.org/download.html.

3.4.1.6 OpenGL

OpenGL, tal como se define en Introducción a OpenGL, es el estándar de los gráficos de alto rendimiento. OpenGL es la librería encargada del renderizado de la simulación en tiempo real del entorno de trabajo iluminado. Gracias al modulo de OpenGL de Qt es posible desarrollar aplicaciones con gráficos 3D de una manera rápida y sin complicaciones ni preocupaciones sobre la instalación de las librerías que ya vienen incluidas con Qt.

Documentación, soporte, foros y demás pueden encontrarse en su sitio web http://www.opengl.org/.

3.4.2 Compilación

ZonalCavs puede ser compilado en una gran variedad de sistemas Windows, Linux y Mac. El proceso de compilación en cada plataforma no varía mucho, pero a continuación se explica detalladamente para cada tipo de sistema.

3.4.2.1 Microsoft Windows

Para compilar ZonalCavs en Windows (2000/XP/Vista) es necesario tener instalado los siguientes paquetes:

- Qt Open Source Edition versión 4.3.4
- Mingw32 3.8

Además es necesario tener hardware compatible respecto a la aceleración 3D de video. Lo recomendable es tener al menos 32 Mb de memoria de video con soporte de aceleración 3D. Las tarjetas nVidia o Ati son las más comunes y las que tienen el mejor rendimiento y soporte para OpenGL.

La instalación correcta de la tarjeta de video esta fuera del ámbito de este documento, por lo tanto si existen inconvenientes al momento de compilar o ejecutar el programa, por favor consulte el manual de su tarjeta de video o visite el sitio del fabricante.

La instalación de Qt Open Source Edition y Mingw para Windows no tiene inconvenientes, ya que generalmente vienen con un asistente de instalación y configuración.

Una vez instalado Qt y el compilador Mingo, simplemente hay que abrir una consola de comandos (Inicio-Ejecutar-cmd) y dirigirse dentro del directorio de ZonalCavs que contenga los archivos fuente. Si por alguna razón no existiese el archivo **ZonalCavs.pro** es necesario crearlo, simplemente hay que ingresar el comando **qmake –project** en la consola y luego editar el archivo generado y añadir las siguientes líneas:

```
QT += opengl
QT += sql
win32 {
RC_FILE     = icon.rc
}
```

Luego de guardar las modificaciones al archivo **ZonalCavs.pro**, procedemos a compilar el proyecto con el comando **qmake**, luego de esto simplemente se ingresa el comando **make** y el programa se habrá compilado y su ejecutable se encontrará en el directorio “release”.

Es importante recalcar que para poder distribuir el programa compilado es necesario además distribuir las librerías de enlace dinámicas (DLL) requeridas por el programa para poder ejecutarse en sistemas que no tengan Qt previamente instalado.

Los archivos .dll requeridos dentro del directorio de la aplicación son los siguientes:

- mingwm10.dll
- QtGui4.dll
- QtSql4.dll
- QtOpenGL4.dll
- QtCore4.dll
- /sqldrivers/qsqlite4.dll

Nótese que el último dll esta contenido en un directorio llamado sqldrivers. Esta estructura es necesaria para evitar problemas de enlace de las librerías con el programa principal.

Este proceso de compilación es exclusivo para sistemas Windows de 32 bits. La compilación del programa para plataformas de 64 bits está a manos del lector y podría ser posible usando los mismos pasos pero utilizando un sistema base, software de desarrollo y hardware de 64 bits.

3.4.2.2 GNU/Linux

El proceso de compilación en sistemas GNU/Linux es muy similar al proceso de compilación en sistemas Windows, pero a continuación se describe el proceso de manera especifica.

Para compilar ZonalCavs en GNU/Linux es necesario tener instalado los siguientes paquetes con sus respectivas dependencias:

- Qt Open Source Edition versión 4.3.4
- g++

Además es necesario tener hardware compatible respecto a la aceleración 3D de video. Lo recomendable es tener al menos 32 Mb de memoria de video con soporte de aceleración 3D. Las tarjetas nVidia o Ati son las más comunes y las que tienen el mejor rendimiento y soporte para OpenGL.

La instalación correcta de la tarjeta de video esta fuera del ámbito de este documento, por lo tanto si existen inconvenientes al momento de compilar o ejecutar el programa, por favor consulte el manual de su tarjeta de video o visite el sitio del fabricante.

La instalación de Qt Open Source Edition y g++ para GNU/Linux no tiene inconvenientes, ya que generalmente puede ser encontrado en los repositorios de la distribución que se este utilizando o mediante la compilación de sus fuentes. Personalmente creo que la manera más sencilla es instalar estos paquetes y sus dependencias a través de un gestor de paquetes como **Aptitude** de Debian/Ubuntu, **Yast** de SuSe o **Yum** de Fedora.

Una vez instalado Qt y el compilador g++, simplemente hay que abrir una terminal de comandos y dirigirse dentro del directorio de ZonalCavs que contenga los archivos fuente. Si por alguna razón no existiese el archivo **ZonalCavs.pro** es necesario crearlo, simplemente hay que ingresar el

comando **qmake –project** en la consola y luego editar el archivo generado y añadir las siguientes líneas:

```
QT += opengl
QT += sql
```

Luego de guardar las modificaciones al archivo **ZonalCavs.pro**, procedemos a compilar el proyecto con el comando **qmake**, luego de esto simplemente se ingresa el comando **make** y el programa se habrá compilado y su ejecutable se encontrará en el directorio “release” o en el directorio actual de compilación.

La distribución del programa compilado dependerá de la instalación previa del entorno de ejecución de Qt y sus dependencias. Los requerimientos de ejecución se muestran más adelante tanto para sistemas Windows como para sistemas GNU/Linux.

Este proceso de compilación es exclusivo para sistemas GNU/Linux de 32 bits. La compilación del programa para plataformas de 64 bits está a manos del lector y podría ser posible usando los mismos pasos pero utilizando un sistema base, software de desarrollo y hardware de 64 bits.

3.5 Instalación

ZonalCavs es distribuido de varias formas dependiendo de la plataforma de ejecución.

3.5.1 Microsoft Windows

Para su ejecución en Microsoft Windows, ZonalCavs es distribuido como un paquete de instalación el cual guiará al usuario paso a paso en la instalación del programa en su ordenador.

Los requerimientos mínimos de hardware para la ejecución de ZonalCavs en sistemas Windows son los siguientes:

- Procesador de 500 Mhz
- Disco duro 30 Mb de espacio libre en la partición de instalación
- Memoria RAM 128 Mb
- Memoria de Video 32 Mb con aceleración 3D
- Tarjeta de Video con soporte para OpenGL 1.2
- Monitor a Color VGA resolución de 800x600 pixels
- Teclado
- Ratón con rueda de navegación (scroll)

Los requerimientos de software son los siguientes:

- Microsoft Windows XP Home/Professional/Media Center
- Microsoft Windows Vista Home/Premium/Business
- Controladores de video actuales y correctamente instalados

3.5.2 GNU/Linux

Para la ejecución de ZonalCavs en sistemas Linux, la distribución esta disponible como código fuente (.tar.gz, .zip) o como un fichero ejecutable para Debian/Ubuntu. Para la compilación en sistemas Linux véase Compilación.

Los requerimientos mínimos de hardware para ejecutar ZonalCavs en Linux son los siguientes:

- Procesador de 500 Mhz
- Disco duro 30 Mb de espacio libre en la partición de instalación
- Memoria RAM 128 Mb
- Memoria de Video 32 Mb con aceleración 3D
- Tarjeta de Video con soporte para OpenGL 1.2
- Monitor a Color VGA resolución de 800x600 pixels
- Teclado
- Ratón con rueda de navegación (scroll)

Los requerimientos de software son los siguientes:

- Debian GNU/Linux “Etch” o Ubuntu GNU/Linux “Gutsy Gibbon”
- libqt4-core, libqt4-gui, libqt4-sql y sus respectivas dependencias.
- Controladores de video actuales y correctamente instalados.
- Cualquier otro sistema Linux que cumpla con los requerimientos de hardware y software.

3.6 Pruebas

ZonalCavs ha sido probado exitosamente en Microsoft Windows XP Professional (32 bits), Microsoft Windows Vista Ultimate (32 bits), Ubuntu Linux 8.04.1 “Hardy Heron” y Debian 4.0 GNU/Linux “Etch” (32 bits). Para la compilación de ZonalCavs en otras plataformas y sistemas operativos véase el apartado anterior sobre Compilación.

3.6.1 Aplicación de ZonalCavs en la Sala de Grados de la Facultad de Ingeniería en Ciencias Aplicadas FICA de la Universidad Técnica del Norte.

ZonalCavs fue probado en la Sala de Grados de la facultad, previo la medición del nivel de iluminación actual, para demostrar que no se estaban cumpliendo las normas de iluminación mínima para ese local y además comprobar el buen funcionamiento del programa.

3.6.1.1 Medición y cálculo de las iluminancias puntuales y uniformidad de iluminación.

Para realizar la mediación de las iluminancias puntuales de la sala de grados, fue necesario dividir el local en 8 cuadrículas tal como se muestra en la Ilustración 157. Donde p1-p9 son los puntos donde se midieron las iluminancias mediante el uso de un luxómetro digital cortesía del Cuerpo de Bomberos de Ibarra.

Ilustración 156: Sala de Grados de la Facultad.

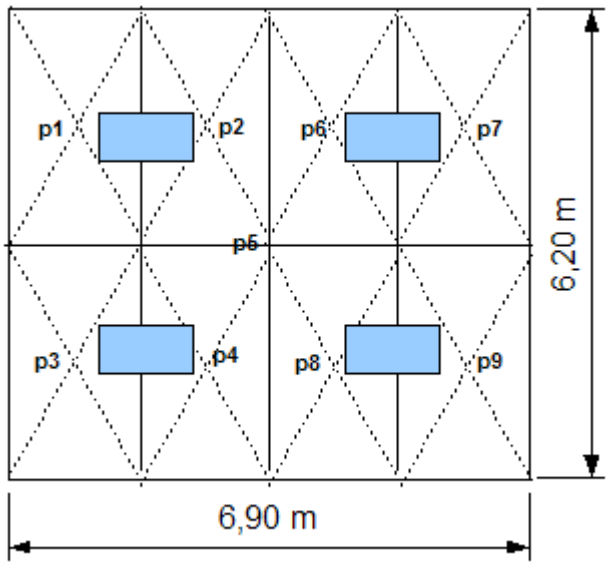


Ilustración 157: Disposición actual de luminarias y división en zonas y puntos de medición.

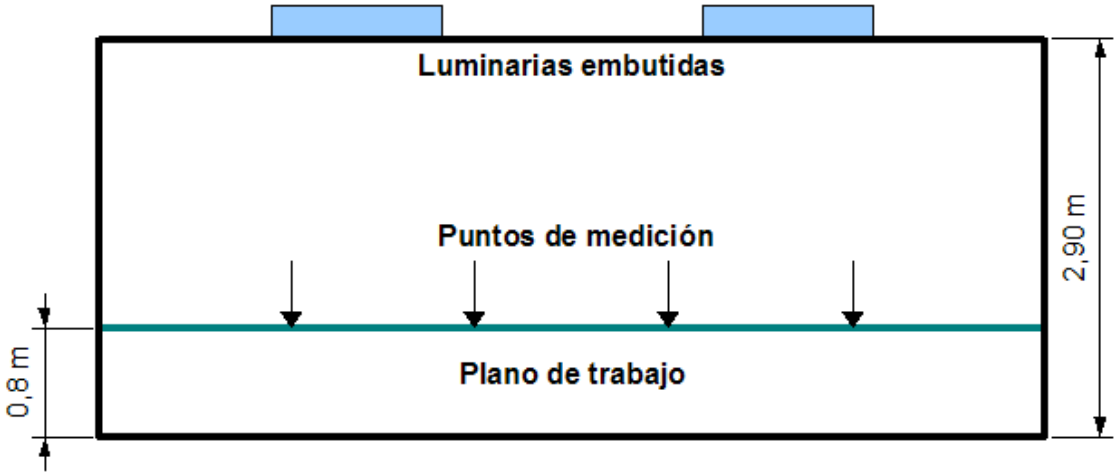


Ilustración 158: Vista frontal del local y puntos de medición.

En la Ilustración 158 se puede apreciar los puntos de medición donde se colocó el luxómetro para obtener la el valor de la iluminancia puntual. Los datos obtenidos y los cálculos respectivos se muestran a continuación.

Tabla 35: Cálculo de la iluminancia promedio y uniformidad de iluminación.

Local: Sala de grados FICA	
Dimensiones: ancho: 6,20 m largo: 6,90 m alto: 2,90 m	
Iluminancia puntual	**Luxes**
E1	75
E2	67
E3	85
E4	91
E5	87
E6	75
E7	83
E8	81
E9	95
Total =	739
$\overline{E}$ =	**82,11**
Emin =	67
Emin (1,3) =	67
Emin (1,2,6,7) =	75
Uniformidad largo =	0,82

Uniformidad ancho =	0,91
Uniformidad Total =	**0,82**

Como se puede apreciar en la tabla Tabla 35, el nivel de uniformidad esta cumpliendo con la normativa de no ser menor a 0.70 pero el nivel de iluminación promedio es muy bajo respecto a los 300 luxes que debería tener una Sala de Grados.

3.6.1.2 Aplicación de ZonalCavs en la sala de grados

Una vez obtenidos los datos respecto al nivel de iluminación actual de la sala de grados, se procede a utilizar el programa para obtener una idea de comó se deberían distribuir las luminarias para obtener el nivel de iluminación adecuado de 300 luxes. Para esto se consultó sobre el tipo de luminarias que estan en uso en el local para obtener la información que requiere el sistema.

La manera más facil de obtener los datos sobre flujo luminoso, potencia, temperatura de color, etc sobre un tipo de luminaria en especial, tal como la utilizada en la sala de grados, es buscar en Internet con información sobre el modelo de la lampara. En este caso, la luminaria es una Sylvannia F40T12/D/DULUXE. Los datos obtenidos desde Internet se listan a continuación:

- Wattage 40
- Lumens 2180
- Base G13
- Shape T12
- Overall Length (mm) 1219
- Life Hours 20000
- Kelvin 6500
- CRI 88

Ilustración 159: Fotografía de la luminaria utilizada en la sala de grados [WEB20].

También es posible determinar de una manera aproximada el flujo luminoso de una luminaria como esta a partir de la siguiente formula:

$$\Phi_v = K.W$$

Donde:

- K es la eficacia luminosa de la luminaria, generalmente 50 para este tipo de lámparas.
- W es la potencia en watts de la luminaria.

Entonces mediante esta formula obtenemos un flujo luminoso de 2000 lúmenes, un valor aproximado que podemos utilizar al realizar cálculos no muy meticulosos.

Luego, la información ingresada al sistema fue la siguiente:

1. **Geometría del local:**
    - Ancho = 6.20 m
    - Largo = 6.90 m
    - Alto = 2.9 m
    - Suspensión = 0.0 (luminarias embutidas)
    - Altura del plano de trabajo = 0.8 m
    - Rotacion de las luminarias = 0º
    - Tipo de local = Local rectangular

2. **Luminaria**
    - Flujo luminoso = 2180 lúmenes
    - Potencia = 40 watts
    - Longitud = 1.22 m
    - Ancho y alto = 0.05 m
    - Temperatura de Color = 6500 ºK
    - Índice de Reproducción de Color = 88
    - Eficacia luminosa (calculada por el sistema) = 54

3. **Materiales y reflectancias**
    - Paredes = pared estándar con 50% de reflectancia
    - Piso = alfombrado con 20% de reflectancia
    - Techo = techo estándar (falso) con 70% de reflectancia

4. **Parámetros del Local**
    - Factor de Mantenimiento = 0.8
    - Nivel de Iluminación = 300 luxes

Los resultados obtenidos son los siguientes:

Tabla 36: Resultados presentados por ZonalCavs.

Iluminación y Distribución		
Iluminación:	350.594	lm
Potencia requerida:	720	w
Número de luminarias total:	3x3=9	u
Uniformidad:	0.855691	
Espaciamiento (ancho):	206.667	m
Espaciamiento (largo):	2.3	m
Reflectancias Efectivas de Cavidad		
Cielorraso (pcc):	70	%
Local/Paredes (pw):	50	%
Suelo (pfc):	19	%
Relaciones de Máximas Luminancias		
Relación Lo/Lf:	2:01	
Relación Lo/Lp:	2:01	
Relación (Lpt+Lpl)/(Lp+Lcr):	208396:1	

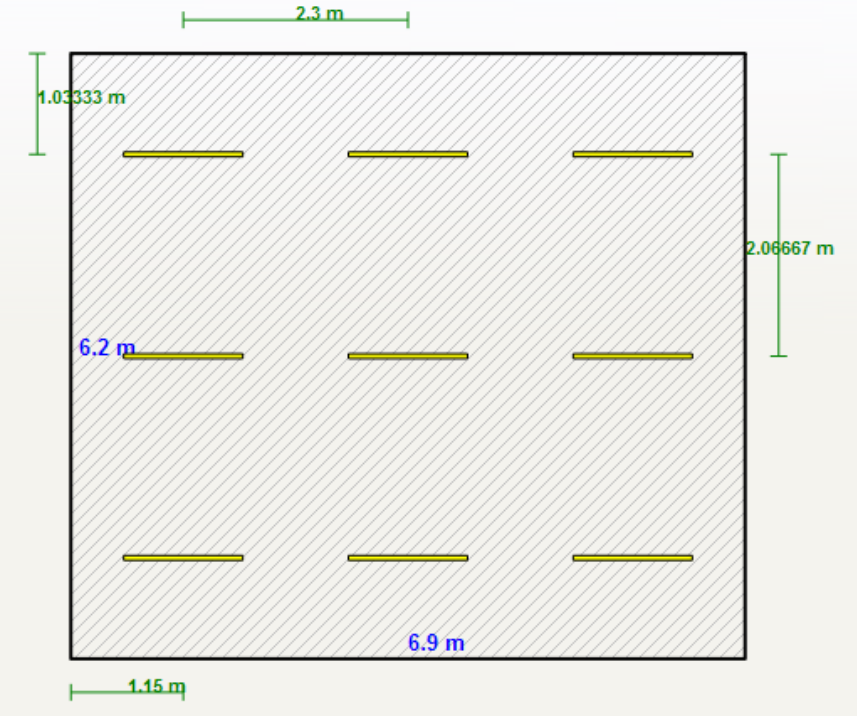


Ilustración 160: Distribución de las luminarias en 2D generada por ZonalCavs.

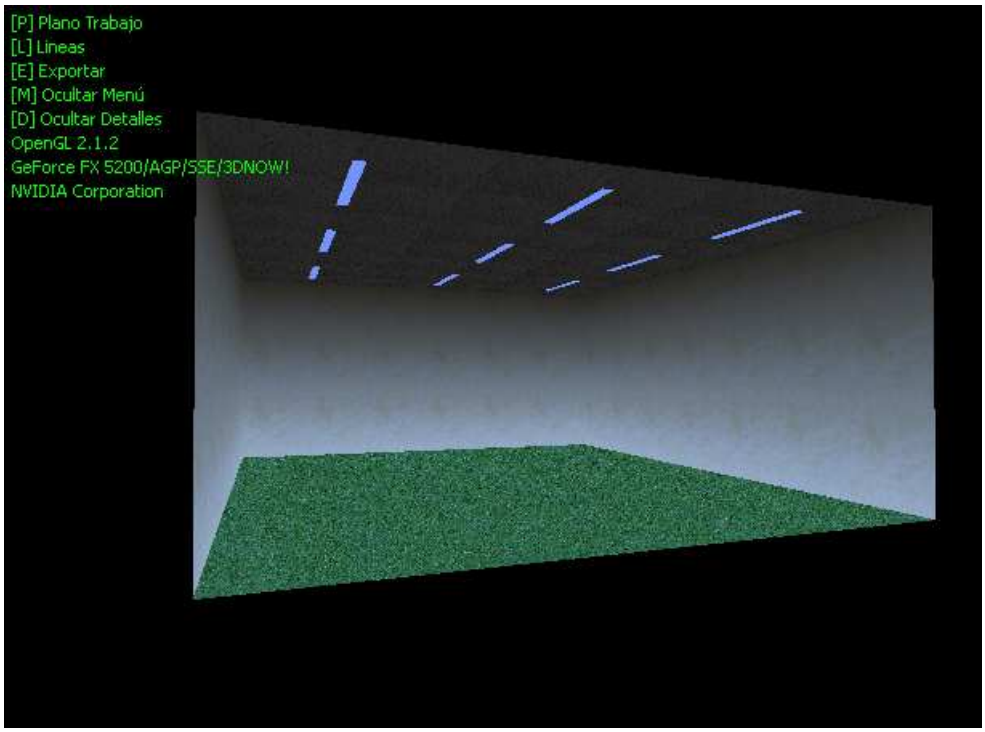


Ilustración 161: Simulación 3D de la sala de grados generada por ZonalCavs.

Como se puede apreciar, ZonalCavs indica que hay que utilizar 9 luminarias Sylvania F40T12/D/DULUXE distribuidas de una manera uniforme para poder obtener el nivel de iluminación deseado.

Como recomendación se puede decir que se podría utilizar luminarias que tengan un mayor flujo luminoso con una potencia baja para de esta manera equilibrar la relación costo/beneficio, es decir, obtener una mayor iluminación con un consumo eléctrico normal o menor y de esta manera ayudar a combatir el Calentamiento Global.

Otras recomendaciones incluyen el evitar las reflexiones molestas dentro de la sala de grados, esto quiere decir, la buena ubicación de proyectores y selección de materiales de fondo (paredes y pizarras para exposición). También hay que considerar los efectos del deslumbramiento, aunque generalmente este tipo de lámparas casi no generan mayores molestias.

3.7 Evolución

ZonalCavs ha cumplido satisfactoriamente los objetivos de esta tesis de investigación al simular la distribución luminosa obtenida mediante los cálculos del Método de las Cavidades Zonales y utilizando el modelo de iluminación de OpenGL (véase Iluminación en OpenGL).

Aun queda mucho trabajo por realizar en este tema y muchas adiciones quedan pendientes para este aplicativo. Muchas de ellas se las puede encontrar en el apartado de Recomendaciones y espero, personalmente, otros investigadores aporten con su experiencia y conocimientos para hacer de este pequeño aplicativo una herramienta indispensable en la lucha sobre el cumplimiento de las normas y precautelar la salud de los empleados y además combatir el Calentamiento Global.

CAPITULO IV

CONCLUSIONES Y RECOMENDACIONES

4 CONCLUSIONES Y RECOMENDACIONES

4.1 Conclusiones

- El modelo de iluminación de OpenGL (véase Iluminación en OpenGL) realiza un muy buen trabajo en la realización de simulaciones de ambientes iluminados, además, brinda al desarrollador la elección del balance entre el rendimiento y la calidad de renderizado de los resultados, además de ofrecer muy buena portabilidad entre plataformas (hardware y software). Especialmente en este trabajo se ha optado por el rendimiento, lo que permite una simulación en tiempo real y una interacción y manipulación del entorno de una manera fluida y rápida.

- El método de las Cavidades Zonales ha sido muy útil en lo referente al cálculo veloz del número de luminarias necesarias para cumplir con un nivel de iluminación requerido en un ambiente de trabajo, pero es importante aclarar que este método esta orientado hacia la iluminación general (véase Métodos de Iluminación) y su comportamiento referente a la iluminación puntual o especifica en un cierto punto esta fuera de sus objetivos. Es por esto que al suspender una luminaria muy cercana al puesto de trabajo (plano de trabajo), los resultados pueden parecer inesperados.

4.2 Recomendaciones

ZonalCavs es un aplicativo que tiene mucho potencial en cuanto a los objetivos y requerimientos que se han planteado, y en el transcurso de su desarrollo han surgido muchas funcionalidades adicionales que podrían ser implementadas en el futuro utilizando técnicas avanzadas de OpenGL u otros métodos de calculo y que se listan a continuación:

- La adición de subrutinas de cálculo que incluyan en la distribución de las luminarias los posibles lugares donde se recomienda ubicar los puestos de trabajo donde el deslumbramiento sea el menor posible. Estos cálculos podrían utilizar el Método de Sollner [DESC99] (véase Confort Visual) o cualquier otro método adecuado.

- Incluir en la simulación en tiempo real objetos premodelados como muebles, sillas, mesas, computadores, etc. para darle más realismo a la simulación y a la vez comprobar la producción de sombras que podrían afectar la productividad del empleado.

- También podría ser útil la adición de cálculos de iluminación puntual en ciertos lugares descritos por el usuario o de manera general y uniforme en varios puntos específicos.

- Otra posibilidad, que quedaría fuera del tema de la utilización de OpenGL como motor grafico, es la utilización de DirectX para realizar las rutinas de renderizado. Lamentablemente con esta opción el sistema seria únicamente para plataformas Windows y se deja el análisis de sus ventajas al lector.

- De manera contraria, queda la posibilidad de la inclusión de técnicas muy avanzadas de OpenGL (que no se incluyeron en este trabajo), por ejemplo shaders de píxel o vértices, tessellators, evaluadores, y muchas más, de manera que la experiencia del usuario durante la simulación sea mucho más agradable. Hay que recordar además, que la utilización de

estas y otras técnicas avanzadas influyen en el rendimiento de la simulación en tiempo real, por lo que probablemente lo mejor sea la producción de una imagen prerenderizada donde la calidad de imagen es casi fotográfica (véase 3D en Tiempo no Real).

4.3 Verificación de la hipótesis

Hipótesis:

Si para desarrollar al máximo las actividades en un sitio de trabajo, se requiere de una buena iluminación y una buena elección del color y sus contrastes, entonces al implementar una aplicación que distribuya de manera eficiente las fuentes de iluminación y simule un entorno virtual aproximado a la realidad, se obtendrá una ayuda invaluable para que las empresas se beneficien de ahorro energético, precautelen la salud visual de sus empleados y ayuden a combatir el Calentamiento Global.

Verificación:

Como se pudo apreciar en el apartado sobre Confort Visual, una buena iluminación influye en el confort visual y laboral de los trabajadores, por lo que una iluminación deficiente en el ambiente de trabajo, que no cumple con las normas de iluminación recomendadas, afecta la productividad del empleado y por consecuencia de la empresa.

Al momento de diseñar la iluminación de un local, se deben considerar varios aspectos importantes en la selección de la luminaria adecuada, por ejemplo la reproducción de los colores, eficacia y eficiencia luminosa, temperatura de color, deslumbramiento, etc. Una selección inteligente de la fuente de iluminación conjuntamente con los cálculos y distribución de luminarias obtenidos de ZonalCavs, producen un diseño de iluminación eficiente, que ayuda a cumplir con los niveles de iluminación recomendados, por lo que la salud y confort de los empleados queda asegurada, además, se ahorra energía y de esta manera se contribuye en la lucha contra el Calentamiento Global tal como se demuestra en [ELIA02]. Entonces, en consideración a estos argumentos la hipótesis planteada se cumple satisfactoriamente.

4.4 Posibles temas de tesis

Como se reviso anteriormente en las recomendaciones, el aplicativo ZonalCavs tiene aun muchas adiciones pendientes, y pueden existir muchas más, por lo que cada una podría ser un tema interesante de tesis. Adicionalmente, es de considerar la excelencia de OpenGL en el desarrollo de simuladores, por lo cual, temas como simulación de fluidos, sistemas de partículas (contaminación del aire), inclusive sobrecarga térmica y ventilación.

Recalcando que el tema de Gráficos por Computadora, y especialmente los gráficos tridimensionales, es muy amplio, y cada una de sus partes puede ser un muy buen tema de investigación. También, orientándose específicamente en OpenGL, las técnicas avanzadas que brinda y que quedaron fuera de este trabajo como por ejemplo el estudio de Evaluadorres y NURBS, Tessellators, Quadratics, píxel y vertex shaders, o el estudio de OpenGL ES pueden ser excelentes e interesantes temas de tesis.

BIBLIOGRAFÍA

Libros

[OPEN07] OPENGL ARCHITECTURE REVIEW BOARD, SHREINER, D. WOO, M. NEIDER, J. y DAVIS, T. 2007. OpenGL(R) Programming Guide: The Official Guide to Learning OpenGL(R), Version 2. 5ta Edición. USA. Addison Wesley. 896 p.

[ASTL04] ASTLE, D. y HAWKINS, K. 2004. Beginning OpenGL Game Programming. 1ra Edición. USA. Course Technology PTR. 336 p.

[WRIG04] WRIGHT, R. S. y LIPCHAK, B. 2004. OpenGL SuperBible. 3ra Edicion. USA. Sams. 1200 p.

[DESC99] DESCHÈRES, L. 1999. Iluminación y Color. Argentina.

[IESN93] IESNA: Illuminating Engineering Society of North America, 1993. Lighting Handbook IES, 8va. Edicion.

[PUEN01] PUENTE, M. 2001. Higiene y Seguridad en el Trabajo. 1ra Edición. Ecuador. 439 p.

[CEI96] CEI, 1996. Aplicaciones Eficientes de Lámparas. Cuadernos de eficacia energética en iluminación Nº 1. Comité Español de Iluminación.

[NAR00] NARENDRAN, N. et al. 2000. “What is useful life for white LEDs?”, IES (Illuminating Engineering Society) Annual Conference, paper Nº 52, Washington, Agust.

[PAN96] PANCHI NUÑEZ, C. E. 1996. Física Vectorial Elemental. Ediciones RODIN, Ecuador, 6ta. Edición. 296p.

Documentos Electrónicos

[ELIA02] E.L.I. Argentina. et al. 2002. Manual de Iluminación Eficiente Argentina.

[BERT00] BERTHELETTE, D. 2000. Enciclopedia de Salud y Seguridad en el Trabajo. Pantallas de Visualización de Datos.

[CHIN98] CHIN, N. et al. 1998. The OpenGL Graphics System Utility Library version 1.3.

Sitios Web

[WEB01] http://es.wikipedia.org/

[WEB02] http://nacc.upc.es/navegacion-aerea/x360.html

[WEB03] http://www.bit-tech.net/hardware/2006/03/20/how_crt_and_lcd_monitors_work/1

[WEB04] http://campusvirtual.uma.es

[WEB05] http://www.educaplus.org/luz/reflexion.html

[WEB06] http://www.gusgsm.com

[WEB07] http://www.ledtronics.com

[WEB08] http://edison.upc.edu/curs/llum/interior/

[WEB09] http://www.arquimaster.com.ar/iluminacion/dilum12.htm

[WEB10] http://www.indalux.es/docs/1/Luminotecnia

[WEB11] http://www.fotonostra.com/grafico/modeloscolor.htm

[WEB12] http://www.mathworks.com/access/helpdesk/help/toolbox/images/f8-15484.html

[WEB13] http://www.procean.no/fishfarms.html

[WEB14] http://www.cprogramming.com

[WEB15] http://www.enlloc.org

[WEB16] http://www.magnifier.com.tw

[WEB17] http://www.gnu.org/philosophy/free-sw.es.html

[WEB18] http://trolltech.com/solutions/application-development

[WEB19] http://www.khronos.org/

[WEB20] http://www.bulbman.com