PONTIFICIA **UNIVERSIDAD CATÓLICA** DEL PERÚ

PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DESARROLLO DE UN PROGRAMA PARA LA PROYECCIÓN CÓNICA DE FIGURAS GEOMÉTRICAS SIMPLES EN R4

Tesis para optar por el Título de Ingeniero Informático, que presenta el bachiller:

Roberto Javier Torres Sovero

ASESOR: Dr. Maynard Kong Wong

Lima, Abril del 2010

Resumen

El presente trabajo de tesis tiene como principal objetivo hacer una extensión de la proyección cónica a R4 implementando un visualizador para gráficos en cuatro dimensiones que muestre la validez de la propuesta.

Geométricamente, la proyección cónica puede interpretarse como la intersección de un cono con un plano. Partiendo de que ambas figuras geométricas pueden ser descritas por ecuaciones vectoriales (que no dependen del número de componentes de dichos vectores), la presente tesis hace un análisis de la proyección cónica en R4, intenta la generalización a Rn de algunos conceptos geométricos utilizados y esboza un diseño de software que permita reutilizar las principales herramientas y buenas prácticas de programación de gráficos en 3D.

Será necesaria la generalización a R4 (y posteriormente a Rn) de algunas figuras geométricas simples como cubos, conos, esferas y planos, no sólo para la construcción de la proyección en sí (a partir de un cono y un plano), sino también para contar con un conjunto de figuras geométricas a dibujar.

En una arquitectura orientada objetos como la propuesta, esto también implica la necesidad de un diagrama de clases de estas figuras geométricas y de un pipeline para el proceso de creación, transformación y proyección de las mismas.

Finalmente, en R4, una clase cámara se encargará de realizar la proyección utilizando transformaciones con matrices y vectores de orden 4, restringiendo la proyección al rango visible de la cámara. Dándole al usuario la posibilidad de mover la cámara en R4 será posible apreciar, en tiempo real, los resultados de las hipótesis desarrolladas en el presente trabajo.

A Honorato y José Mishaja, maestros de la visualización

Quiero agradecer a mi asesor, el Dr. Maynard Kong, por motivarme y valorar mi trabajo y ayudarme a que esta tesis haya sido finalmente aceptada en la Facultad de Ciencias e Ingeniería. También quiero agradecer al profesor José Antonio Pow Sang por su ayuda con los formatos y los trámites necesarios para presentar este documento, a Enrique Mayorga por haberme transmitido desde el principio de la carrera una fascinación muy especial por los números, a Emilio Sovero y a Pedro Otero por motivarme a perseverar en el mundo académico, a Javier Ortiz y a Guillermo Rodríguez por enseñarme C++ y casi todo lo que sé de programación de gráficos, y a Alba Godoy por tentarme con el reto de una generalización a Rn del proyecto. Finalmente, un agradecimiento muy especial a mis padres y a mi hermana por su eterno apoyo incondicional.

Índice

Introducción

La posibilidad de visualizar conceptos a través de gráficos es una importante herramienta para la transmisión de información al observador. Gráficos en 2 y 3 dimensiones son fundamentales para ilustrar relaciones y graficar conceptos en cualquier área del conocimiento. Sin embargo, exceptuando diagramas de redes y animaciones en las que el tiempo pasa a ser una cuarta variable, daría la impresión de que, para relaciones en las que intervienen más de tres variables, no es posible la utilización de gráficos.

Esta tesis busca implementar una extensión a R4 de la proyección cónica para dibujar gráficos en cuatro dimensiones con algunas figuras geométricas simples como cubos, conos, esferas y planos. Dado que la proyección cónica se construye a partir de la intersección de un cono con un plano, la generalización de estas figuras a Rn significará también la generalización a Rn de la proyección misma.

Siendo R4 el espacio de 4 dimensiones definido por 4 vectores linealmente independientes, el principal problema a nivel geométrico será la necesidad de emplear matrices de 4 dimensiones para la traslación, rotación y proyección de los objetos a ser visualizados, con la dificultad adicional de que muchas veces los vectores que componen estas matrices tendrán que ser ortogonales entre sí.

Además, la necesidad de contar con una cámara y con figuras geométricas, implica el desarrollo de clases para objetos en R4 que deberán ser construidos y ubicados en el espacio a partir de sus constructores, propiedades y métodos.

Una vez obtenida la proyección de R4 a R3, las librerías de gráficos en 3D de DirectX y las buenas prácticas de programación recomendadas para animaciones cuadro por cuadro facilitarán el dibujo de la proyección resultante.

Capitulo 1: Generalidades

En este capítulo se tratará el origen del proyecto, la definición del problema, el plan de proyecto y el marco conceptual para entender la proyección cónica en R3 y la necesidad de extender a R4 los conceptos de algunas figuras geométricas para lograr una proyección cónica de R4 a R3.

1.1. Génesis del proyecto

Escribir esta tesis sin mencionar ciertos avances que fui logrando con el transcurso del tiempo y una enorme fascinación por los gráficos, probablemente daría la impresión de un trabajo surgido de manera espontánea, carente del largo proceso del que en realidad forma parte. Por esta razón, en los siguientes subcapítulos se hará una breve descripción de los orígenes del proyecto.

1.1.1. El Inicio

A mediados del año 2000 empecé a hacer gráficos simples con comandos del software Mathematica 3.0 de la Wolfram Research. Al no requerir manejo de memoria ni declaración de tipos de variables y con un lenguaje de programación sencillo, de sintaxis parecida al C, este software permitía dibujar polígonos, curvas y superficies con muy pocos conocimientos de programación.

En el Mathematica 3.0, no hay necesidad de los manejadores de ventanas, ni de los buffers de vértices, ni de las matrices de vista que hacen tan complicado dibujar figuras tan simples como líneas y polígonos regulares al trabajar con C++ y las librerías de DirectX.

En el lenguaje del Matehmatica, en cambio, para dibujar un cuadrado, basta con abrir una nueva hoja de cálculo y escribir

```
Show[Graphics[Polygon[{ {1,1},{1,-1},{-1,-1},{1-,1} }]]]
```

y evaluar la sentencia. El Mathematica devolverá entonces una imagen con el cuadrado dibujado.

Una forma más parametrizada de describir un cuadrado es utilizando un comando **Table** para generar los puntos que conforman los vértices del polígono con un parámetro i:

```
Show[Graphics[Polygon[
        Table[{Cos[i*2Pi/4], Sin[i*2Pi/4]}, {i, 0, 3}]
        ]]]
```

Sin embargo, hay que tener en cuenta que, con ésta nueva sentencia, los vértices habrán rotado 45 grados y estarán sobre los ejes x e y como se muestra en la figura 1.1.1-1:

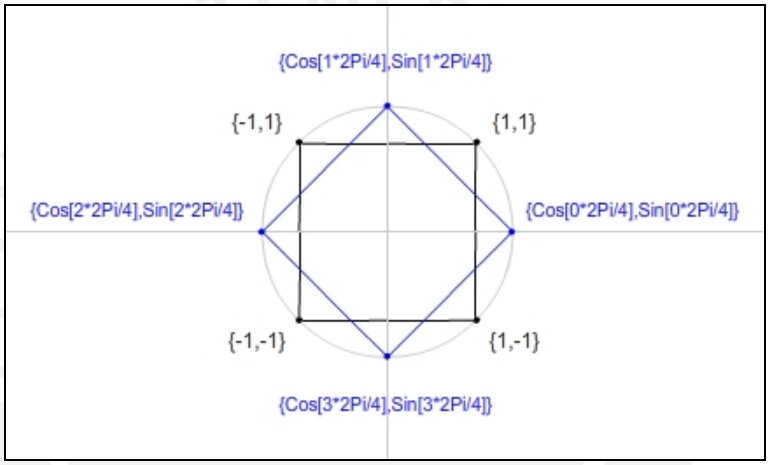


Figura 1.1.1-1

La gran ventaja de ésta parametrización es, que para hacer rotar al polígono sobre su eje, bastará con agregar un parámetro t dentro de las sinusoidales y copiar la sentencia anterior dentro de un bucle **While** o **For** o dentro de un comando **Table**:

```
t=0;
While[ t<2Pi,

        Show[Graphics[Polygon[
                Table[{Cos[i*2Pi/4 + t], Sin[i*2Pi/4 + t]}, {i, 0, 3}]
                ]]];

        t += 2Pi/100
]
```

Esto genera una secuencia de gráficos con t incrementándose de 0 a 2Pi, haciendo dar al polígono una vuelta completa. Luego, con hacer doble click sobre cualquiera de los gráficos generados, se muestra una animación.

Así empezó mi etapa de programador de gráficos, dibujando polígonos de colores que describían distintos giros en el tiempo.

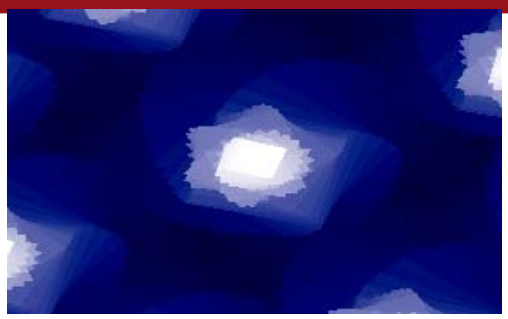


Figura 1.1.1-2

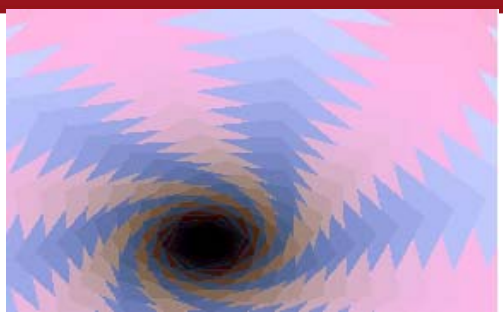
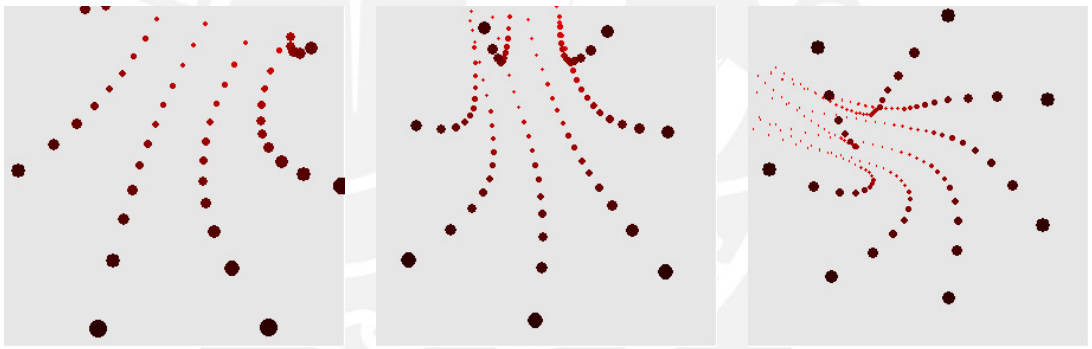


Figura 1.1.1-3

A finales del 2001 presenté un video con animaciones en la exposición de fin de año la Facultad de Arte. A principios del 2002 empecé a hacer gráficos en tres dimensiones con el comando **Show[Graphics3D[ ]]** del Mathematica 3.0. Sin embargo, limitaciones para controlar el rango visible y la posición del observador me obligaron pronto a volver al comando de gráficos bidimensionales **Show[Graphics[ ]]** y trabajar las figuras en R3 como elementos bidimensionales transformados previamente por una función 'Camara'. Por esta época utilicé por primera vez una curva de Lissajou aprovechando la ortogonalidad de sus derivadas para dibujar puntos en el espacio y crear una animación que simulaba un viaje dentro de un túnel.



Entonces apareció la necesidad de abandonar el Mathematica 3.0 y empezar a ejecutar animaciones en tiempo real, programadas en C++, con las librerías de DirectX y con el Microsoft Visual Studio 6.

1.1.2. Animaciones 3D en tiempo real

A finales del 2001, unos amigos empezaron a hacer música con Reaktor, un software modular de la Native Instruments para sintetizar sonidos. Enrique Mayorga, entonces estudiante de ingeniería electrónica, tenía una propuesta bastante original: generar frecuencias de sonido y ritmo que estuvieran relacionadas entre sí por números naturales pequeños. Era una apuesta fuerte por lo que en música se conoce como 'Afinación pitagórica'. La posibilidad de multiplicar y dividir frecuencias reiteradas veces vía software, abría un amplio espectro de posibilidades que trascendía la capacidad de los instrumentos físicos tradicionales. Nuevas formas de instrumentos para recorrer dicho espectro empezaron a brillar por su ausencia.

En la primera mitad del 2002, Enrique Mayorga formó un grupo de estudiantes de ingeniería electrónica para construir un instrumento musical laser que pudiese

interactuar con los módulos de Reaktor vía mensajes MIDI. Teniendo conocimiento de mis animaciones en el Mathematica y habiendo musicalizado con afinación pitagórica el video presentado en la exposición de arte a finales del 2001, Enrique me propuso controlar las animaciones en tiempo real vía el instrumento láser. Yo acepté.

Para cuando empecé a leer sobre C++ y programación orientada a objetos (por entonces sólo sabía Pascal), el proyecto ya había sido presentado al ISEA 2002 (International Symposium on Electronic Art), a realizarse en la ciudad de Nagoya, Japón, y había sido seleccionado.

Javier Ortiz, un estudiante de informática con experiencia en programación de juegos que cursaba los últimos ciclos de la carrera, se integró al grupo para ayudarme a traducir los algoritmos del Mathematica a C++ y para habilitar la comunicación por el puerto MIDI. Desde entonces tengo la costumbre de probar cualquier idea primero en el Mathematica y luego traducirla a C++ o C#, aunque es cierto que, actualmente, hay herramientas sumamente útiles que se encuentran a mitad de camino entre ambos extremos como Processing (basado en java) y el ActionScript de Flash.

En cuanto al proyecto del arpa láser, finalmente, los objetivos se lograron, el instrumento interactuaba con módulos de Reaktor que generaban audio en afinación pitagórica y controlaban algunos parámetros de las animaciones, hicimos un par de exposiciones, conseguimos el financiamiento para viajar a Nagoya y participamos del simposio exponiendo un video sobre el proyecto y publicamos un *paper* titulado *'Interactive system for exploring audiovisual harmony'*.

Para entonces ya no hacía más animaciones bidimensionales: en el nuevo entorno de programación que ejecutaba animaciones en tiempo real, todas las coordenadas tenían tres componentes, era necesario trabajar con matrices y había logrado implementar una clase 'Camara'.

Considero una pena que el proyecto no diera para más, en parte, porque para la dirección académica de investigación de la universidad, el éxito consistía en vender masivamente el instrumento en la próxima campaña navideña, algo que, para algunos integrantes del grupo, sumergidos en profundos conceptos de armonía y geometría, no venía al caso.

1.1.3. La visualización de un mundo oculto a través de ventanas

Cuando volví a la programación de gráficos, luego de retirarme un par de años de la universidad, intenté hacer animaciones en red. Me obsesioné con la idea de un mundo complejo, una realidad paralela accesible sólo a través de los monitores sincronizados que la dibujaban desde varias perspectivas, algo parecido a lo que sucede en los juegos en red con varios usuarios.

Sin conocimientos de redes, tuve muchas complicaciones con la comunicación y sincronización de las distintas animaciones corriendo en computadoras distintas. Leyendo sobre geometría y politopos se me ocurrió que el mundo a visualizarse fuese de cuatro dimensiones.

Encontré una notación para sólidos multidimensionales, el 'Símbolo de Schläfli', desarrollada por el matemático suizo Ludwig Schläfli en el siglo XIX para politopos en n dimensiones. Nunca llegué a utilizar esta notación pero tener conocimiento de

ella me dio la certeza de que había figuras geométricas análogas a los sólidos platónicos tridimensiones en R4 y en Rn.

El siguiente paso fue pensar en una cámara que pudiese 'filmar' en este mundo de 4 dimensiones. Entonces, al igual que cuando intentaba proyectar figuras 3D en gráficos bidimensionales con el Mathematica 3.0, volví a recurrir a las curvas de Lissajou, primero buscando una parametrización esférica en R4 y, luego, comprobando que todas las derivadas fuesen ortogonales entre sí (calculando el producto punto entre ellas). Una vez logrado esto, el proceso se convirtió en una iteración más de lo que ya había hecho anteriormente para proyectar coordenadas de R3 a R2.

Éste último paso es el eje central del presente trabajo de tesis que describe el marco conceptual de toda la problemática para poder trasladar y rotar una cámara (y diferentes figuras geométricas) en R4. Una generalización a Rn también es tratada de manera superficial.

1.2. Estado del arte

En internet se trata el tema de la cuarta dimensión como un área muy prometedora tanto para las artes plásticas, como para el entretenimiento y la investigación en general. Llegué a leer algunos artículos en los que se mencionaba incluso un inminente boom del término '4D' luego del rotundo éxito alcanzado por el cine y la televisión en tres dimensiones (ambos aun en etapa de desarrollo).

Sin embargo, a pesar de que la industria de los gráficos por computadora se ha convertido en una de las más rentables y es pieza fundamental en la industria del entretenimiento, no pude encontrar productos de gráficos multidimensionales orientados al usuario final, sólo algunas páginas web que trataban el tema de la geometría multidimensional y algunos videos en youtube con animaciones muy interesantes desarrolladas por programadores interesados en el tema.

1.2.1. Matrices de orden 4 en la programación de gráficos

En el mundo de la programación de gráficos, en cambio, la utilización de matrices de orden 4 para la transformación de puntos es tan común, que los ejemplos más sencillos de DirectX y OpenGL las utilizan para generar animaciones simples. El uso de cuaterniones también está bastante difundido y librerías con cuaterniones y operadores entre cuaterniones vienen incluidas en las librerías por defecto de DirectX.

El éxito de estas herramientas se debe a que la performance de la ejecución de animaciones en tiempo real se incrementa notablemente al dividir las transformaciones en varias multiplicaciones simples, lo que ocurre al trabajar con matrices y cuaterniones.

La nueva apuesta de la mayoría de productores de tarjetas de video, es optimizar aun más la ejecución de estos cálculos procesándolos en paralelo. Un excelente ejemplo es la arquitectura de computación en paralelo CUDA desarrollada por Nvidia para sus unidades de procesamiento de gráficos.

1.2.2. El símbolo de Schläfli

En lo teórico, considero muy positivo haber leído sobre el trabajo de politopos de Ludwig Schläfli, matemático suizo del siglo XIX que, junto con Arthur Cayley y

Bernhard Riemann, es considerado uno de los arquitectos fundamentales de la geometría multidimensional.

Aunque el término 'politopo', creado por Alicia Boole (hija del matemático y filósofo irlandés George Boole), se refería inicialmente a la generalización a Rn de polígonos bidimensionales y poliedros tridimensionales, el concepto puede extenderse a todo lo que exista en el plano y en el espacio tridimensional.

La idea de politopo es de suma importancia para poder contar con una gama de figuras geométricas a ser proyectadas por una cámara en Rn. Más aun, la cámara en sí puede ser considerada también un politopo, dado que, finalmente, no es más que un cono que intersecta líneas de proyección con un plano perpendicular.

El gran aporte de Schläfli en cuánto a politopos radica en la nomenclatura que lleva su nombre. El 'símbolo de Schläfli' se define como {p,q}, si las caras del politopo son *p-gonos* y cada vértice está rodeado por *q* caras. Para la tercera dimensión, {3,3} representa un tetraedro, así como {5,3} representa un dodecaedro. Por ejemplo, {5,3} se interpreta como: *politopo de caras pentagonales que tiene 3 pentágonos uniéndose en cada vértice.*

Lo interesante es la 'escalabilidad' del símbolo de Schläfli para politopos en 4 o más dimensiones. Un hipercubo, por ejemplo, se representa por {4,3,3} y se interpreta como un politopo de faceta {4,3} (un cubo), con 3 facetas coincidiendo por vértice.

En la generalización, un politopo con símbolo de Schläfli {p1, p2, .., pn-1} se define recursivamente como un politopo de faceta {p1, p2, .., pn-2} y politopo {p2, p3, .., pn-1} en cada vértice.

Curiosamente, la mayoría de imágenes de politopos que encontré en la red utilizan la proyección paralela, quizá por resaltar la simetría de las figuras o porque dicha proyección es más sencilla de utilizar en dimensiones mayores a R3.

1.2.3. En el arte

En el campo de las artes plásticas, pintores de la talla de Joan Miró y Salvador Dalí intentaron llevar por primera vez el tema de la cuarta dimensión a la pintura. Joan Miró fue, en este sentido, el precursor de esta iniciativa, escribiendo textos sobre pintura de cuatro dimensiones y experimentos con esculturas en vidrio.

Salvador Dalí, por su parte, pintó cuadros como 'Corpus Hypercubus' (un Cristo crucificado en una cruz construida a partir de 8 cubos) y 'En busca de la cuarta dimensión', donde aparecen relojes derretidos y algunos dodecaedros.

La considerable cantidad de videos en youtube con transformaciones en cuatro dimensiones invita a pensar que estas primeras iniciativas tienen aun mucho camino por recorrer.

En literatura, H.P Lovecraft (Estados Unidos 1890-1937), maestro del género de terror, relata en su cuento 'Los sueños en la Casa de la Bruja' la historia de un estudiante de matemáticas de la universidad de Miskatonic, Walter Gilman, que decide alquilar una habitación en la Casa de la Bruja, lugar en el que vivió Keziah Mason, una vieja hechicera que escapó de los tribunales de Salem en el año 1692 gracias a su habilidad para traspasar dimensiones mediante el trazado de complicadas líneas y curvas en las paredes.

Gilman, sumamente interesado en la cuarta dimensión y el espacio no euclidiano y enormemente cautivado por la misteriosa habilidad de la bruja, decide profundizar sus investigaciones matemáticas en la habitación de Keziah, sin imaginar que será presa de los más terribles sueños mezclados con realidad, que lo llevarán a una muerte horrible e inimaginable.

1.3. Geometría de la proyección cónica en R3

La proyección cónica es quizá la proyección de perspectiva más realista de todas las que existen ya que logra simular el efecto de la profundidad del espacio y la deformación de los ángulos. Al trazar todas las líneas de proyección desde un mismo punto (la posición del observador), la imagen resultante se puede entender como la intersección de un cono con un plano ortogonal a este.

Para generar la proyección se debe conocer la posición del vértice del cono, así como la dirección de su eje y el ángulo de abertura que restringe el campo visual. Además es necesario un vector 'Arriba' que señale el eje vertical del plano de proyección y defina implícitamente al eje horizontal como el producto cruz del vector 'Arriba' por el eje del cono. La orientación de la imagen final, dónde es 'arriba', 'abajo', 'izquierda' y 'derecha', dependerá entonces del vector ‘Arriba’.

La secuencia de pasos para llegar a proyectar los objetos tridimensionales dentro de un rango visible a un plano de proyección puede ser realizada por una clase 'Camara'. Un método de la clase debe encargarse de configurar una matriz de proyección y multiplicar cada vértice de los objetos en el espacio por dicha matriz. Esta transformación consiste, por lo general, en trasladar el origen de coordenadas al vértice del cono y alinear al nuevo eje Z con el eje del cono. Se dice entonces que los puntos han sido transformados a ‘coordenadas de vista’. Luego es mucho más sencillo proyectar los puntos dentro del rango visible al plano XY o a un plano paralelo. El vector 'Arriba' se encargará de definir la dirección de los ejes X e Y en el plano de proyección.

```
class Camara
{
    Vector3 vPosicion;  // Punto en el que se encuentra la cámara (vértice del cono)
    Vector3 vHacia;     // Punto hacia donde mira la cámara
    Vector3 vArriba;    // Vector que define el eje vertical del plano de proyección

    float Angulo;

    void Transformar()
    {
      ...
     }
}
```

El método 'Transformar' necesita de todas las propiedades de la clase para poder configurar la matriz de vista ya que la transformación del punto P se calcula como:

$$P' = (P - vPosicion) \cdot \begin{pmatrix} Ax & Bx & Cx \\ Ay & By & Cy \\ Az & Bz & Cz \end{pmatrix}$$

donde A, B y C son los nuevos ejes de coordenadas con el nuevo origen trasladado al vértice del cono

C = vHacia - vPosicion / || vHacia - vPosicion || (eje del cono, objetivo)

B = vArriba / || vArriba || (eje vertical del plano de proyección)

A = B x C

Luego, el nuevo conjunto de todos los puntos P' (en coordenadas de vista) debe ser proyectado sobre un plano de proyección paralelo al plano XY. La componente Z de este plano es una constante **d** (por lo general igual a 1) que es necesario especificar para conocer la distancia a la que se encuentra el plano de proyección del nuevo origen de coordenadas.

Calcular ahora el punto P" en el que las líneas de proyección trazadas desde el vértice del cono hacia P' cortan al plano de proyección, se reduce a un problema de semejanza de triángulos...

$$P'' = \frac{d}{Pz' * \mathrm{Tan}[\theta/2]} \{Px', Py'\}$$

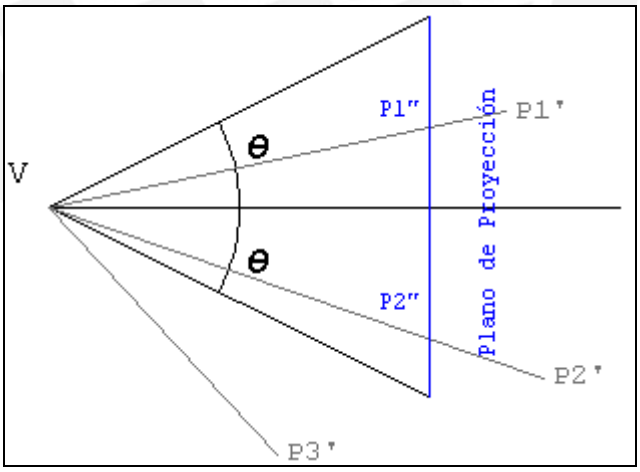


Figura 1.2.3-1

Se debe tener en cuenta que, dado que A es igual a B x C y que A es uno de los ejes de coordenadas del nuevo sistema, el vector C y el vector B tienen que ser siempre perpendiculares entre sí, de lo contrario, los nuevos ejes de coordenadas no serán linealmente independientes y la imagen proyectada aparecerá distorsionada. Una forma de hacer el algoritmo más seguro y garantizar la ortogonalidad entre B y C es haciendo que el vector B sea:

B = vArriba x C/ || vArriba x C ||

producto cruz de vArriba por el eje del cono. Ahora, aun así vArriba y C no sean perpendiculares, B y C sí lo serán.

1.4. El triedro móvil y las curvas de Lissajou

Hemos visto que la clase Camara necesita de los vectores vPosicion, vHacia y vArriba para poder ubicarse en el espacio, generar un triedro, transformar los puntos del sistema original a coordenadas de vista (con el triedro como nuevo sistema de coordenadas) y generar luego la proyección tomando en cuenta el rango visible, es decir, el ángulo del cono cuyo eje es paralelo al vector vHacia.

Por lo tanto, para lograr un movimiento de cámara continuo lo ideal sería contar con una función que describa una trayectoria en el espacio y pueda devolver, para cada punto de la curva, tres vectores unitarios perpendiculares entre sí. Una opción es tomar una curva de clase C2 y formar el triedro a partir de sus derivadas. Por ejemplo, la función

F[t,A,B] = { Cos[A t] Sin[B t], Sin[A t] Sin[B t], Cos[B t] }

que representa una curva de Lissajou en el espacio si A y B son constantes naturales. Por las sinusoidales en sus componentes, la curva tiene infinitas derivadas y está contenida en una esfera de radio igual a 1 ya que

O[u,v] = { Cos[u] Sin[v], Sin[u] Sin[v], Cos[v] }

es la parametrización de una superficie esférica en R3. De hecho, una curva de Lissajou en R3 puede expresarse como:

F[t,A,B] = O[t * A, t * B]

Si tomamos las dos primeras derivadas de F

F1[t,A,B] = { B Cos[A t] Cos[B t]-A Sin[A t] Sin[B t],
B Cos[B t] Sin[A t]+A Cos[A t] Sin[B t],
-B Sin[B t] }

F2[t,A,B] = { -2 AB Cos[Bt]Sin[At]-AA Cos[At]Sin[Bt]-BB Cos[At]Sin[Bt],
2 AB Cos[At]Cos[Bt]-AA Sin[At]Sin[Bt]-BB Sin[At]Sin[Bt],
-BB Cos[Bt]}

y hacemos

F3[t,A,B] = F1[t,A,B] x F2[t,A,B]

se cumplirá que F1, F2 y F3 serán ortogonales entre sí y que u1 = F1 / ||F1||, u2 = F2 / ||F2|| y u3 = F3 / ||F3||, formarán un triedro en cada punto de F.

Multiplicando F por un factor de escala s = 3 y tomando los vectores unitarios de F1, F2 y F3 evaluados en t = Pi / 2, A = 2 y B = 1, se obtiene el siguiente gráfico en el que u1, u2 y u3 aparecen dibujados en azul, verde y rojo respectivamente. En este caso, la curva está contenida dentro de una esfera de radio 3.

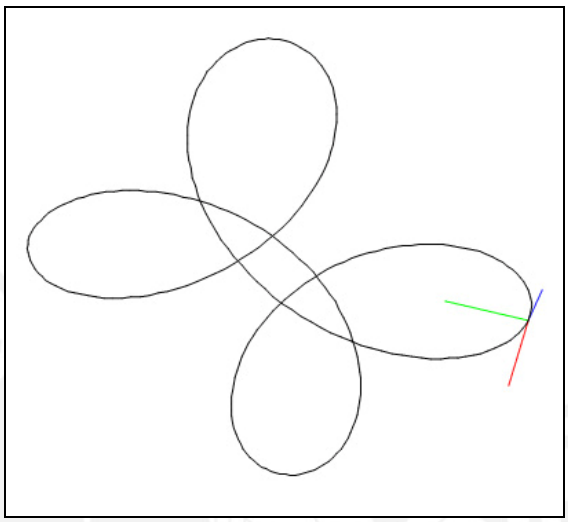


Figura 1.2.3-1

Como la curva se cierra cuando t es igual a 2π, este intervalo se puede partir en n segmentos. Dibujando un triedro dando saltos en t de 2π / n. Para n = 1000, se obtiene la siguiente imagen:

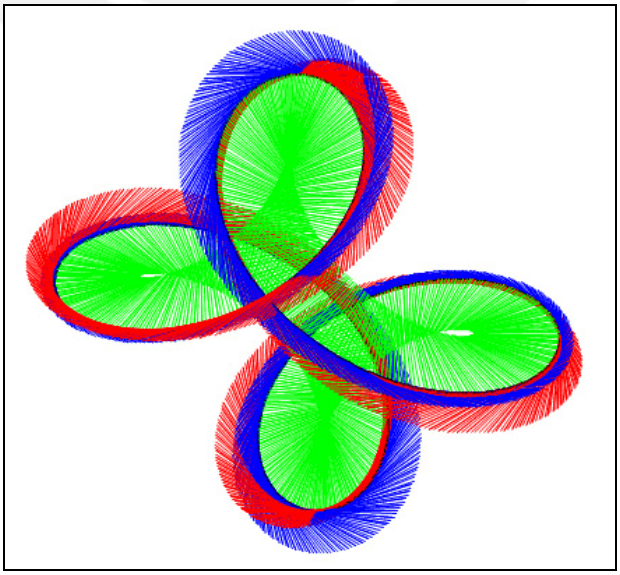


Figura 1.2.3-2

Esto era lo que se necesitaba para lograr un movimiento continuo de la cámara, una curva que sirva de 'carril' y a partir de la cual se pueda obtener tres vectores unitarios perpendiculares entre sí para definir el encuadre.

Con estas propiedades, en cada punto de F, es muy sencillo dibujar una figura en el plano perpendicular a la tangente ya que los vectores unitarios u2 y u3 (verde y rojo) que determinan el plano de proyección sirven a la vez como ejes X e Y. Por ejemplo, para dibujar un círculo en este plano bastará con graficar la curva

$$h[\theta, r] = F[t,A,B] + \{ F2[t,A,B], F3[t,A,B] \} . \{ Cos[\theta], Sin[\theta] \}*r$$

con θ de 0 a 2π y la variable r como radio de la circunferencia. Las variables t, A y B son constantes que determinan el punto de F donde se dibujará el triedro (donde se pondrá el vértice del cono). En el gráfico se muestra ahora una circunferencia de radio 1 contenida en el plano determinado por u2 y u3.

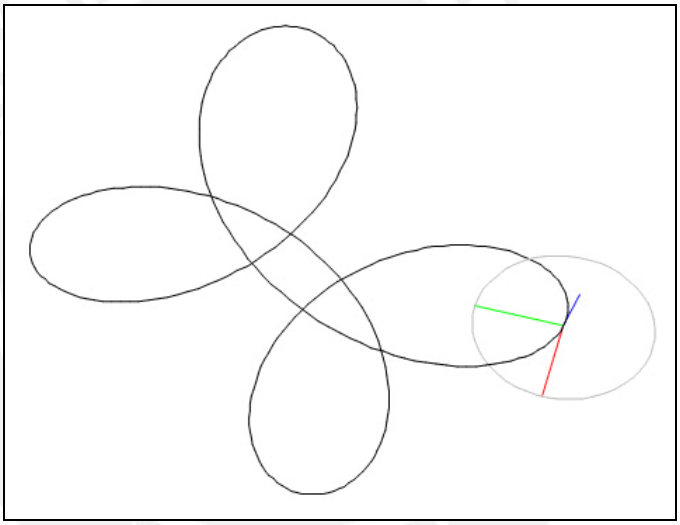


Figura 1.2.3-3

Este círculo puede dibujarse en distintos puntos de F y tendrá siempre a F1 (azul) como vector normal. Si se dibujan polígonos regulares en vez de círculos y se unen todos los vértices de todos los polígonos en la curva, se obtiene la siguiente figura:

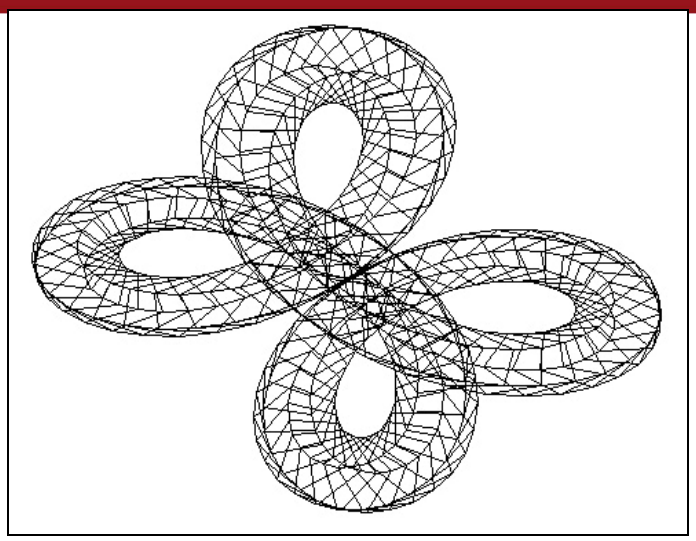


Figura 1.2.3-4

A partir de este modelo tridimensional es posible visualizar el recorrido de una cámara que viaja por F y mira siempre hacia F1, teniendo a F2 y F3 como vectores que determinan el plano de proyección. La animación resultante mostraría un recorrido por el interior del túnel, que no tendría ni principio ni fin.

Las tablas 1.2-1 y 1.2-2 muestran un ejemplo de una proyección en la que se utilizan las tres derivadas de una curva de Lissajou, de parámetros A=2 y B=3, para mostrar un recorrido por el interior del túnel de heptágonos.

La primera tabla muestra la curva de Lissajou vista desde afuera mientras que la segunda muestra imágenes del interior de esta.

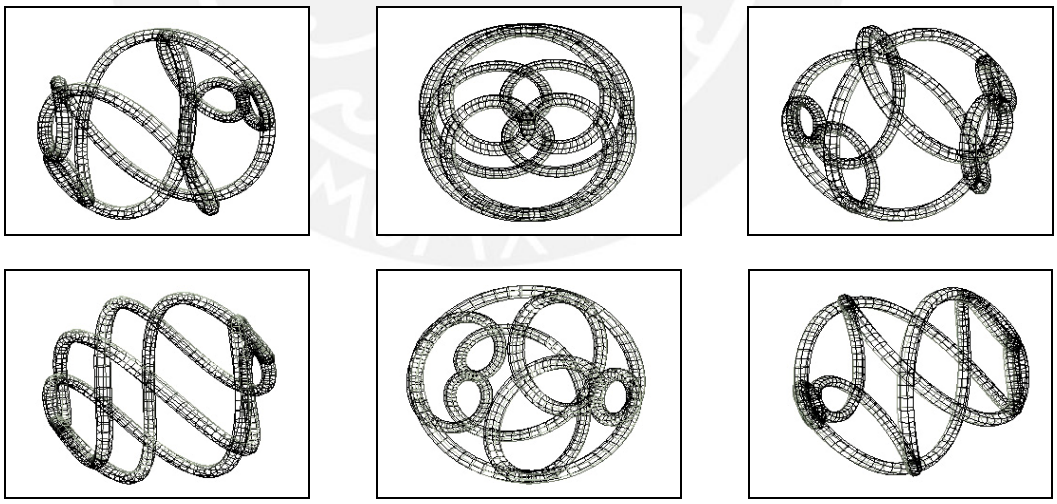


Tabla 1.4-1

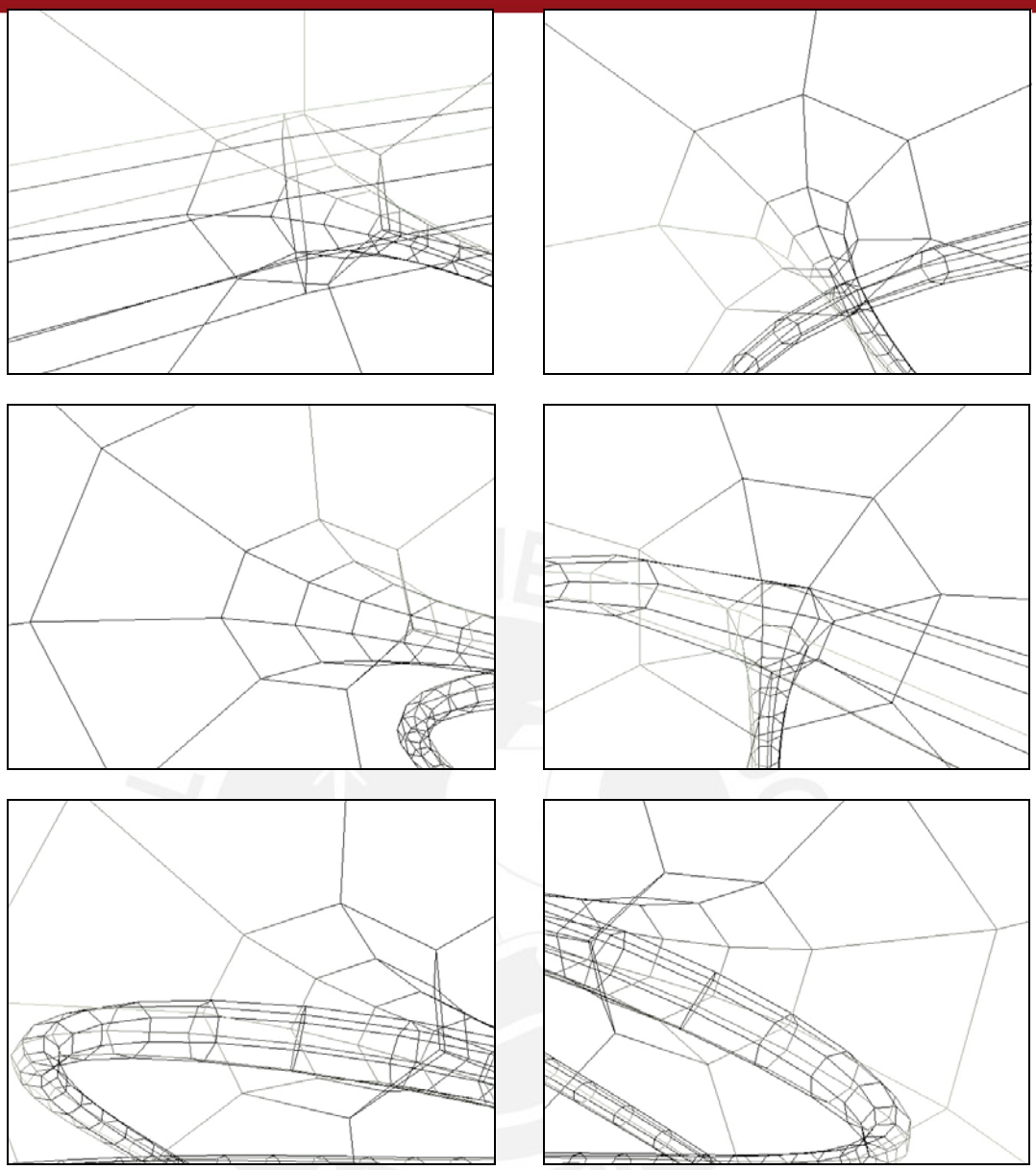


Tabla 1.4-2

1.5. La esfera unitaria y la cuarta dimensión

Como ya se mencionó anteriormente, la imagen que produce una cámara en el espacio tridimensional puede reducirse a la intersección de un cono con un plano. Para dar el paso a la cuarta dimensión es necesario mantener el mismo principio y hacer la misma intersección en R4. Si pensamos tanto en la definición vectorial de un cono como en la de un plano, vemos que estas definiciones pueden ser generalizadas para Rn.

Un cono, por ejemplo, es el conjunto de puntos que satisfacen la desigualdad:

$$\mathbf{(P - vertice) \cdot ejecono \,/\, ||(P - vertice)||\, ||ejecono|| \geq Cos[\,\theta\,]}$$

Es decir, que un punto P está contenido dentro de un cono si el ángulo entre el eje y el vector trazado desde el vértice del cono hasta P (vector VP) es menor que el ángulo θ. Sin embargo, para determinar este ángulo se utiliza el producto punto de ambos vectores, que devuelve siempre un escalar. Y al devolver un escalar, el

producto punto hace que esta condición sea válida independientemente del número de componentes que tengan los vectores y que, por lo tanto, la inecuación pueda ser generalizada para Rn.

En la gráfica se muestra un cono en R2, con los puntos P1 y P2 contenidos en el cono y P3, fuera de él.

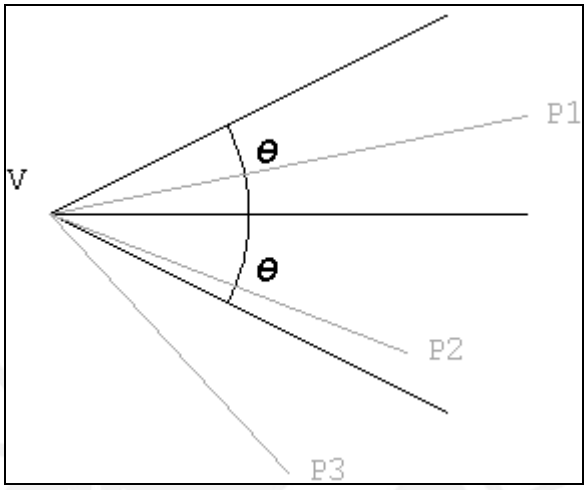


Figura 1.2.3-1

Con el plano de proyección ocurre lo mismo. Nuevamente el producto punto hace que la expresión vectorial que define al plano

(P - PtoPaso) . N = 0

sea válida independientemente de si los vectores están en R2, R3 o Rn. Sólo hay que tener en cuenta que la normal del plano siempre va a poder ser alineada con uno de los ejes del sistema de coordenadas y que, por lo tanto, la proyección va a tener que darse sobre los ejes restantes. Esto implica que una cámara en un mundo en Rn siempre tomará imágenes en n-1 dimensiones, de la misma manera que las cámaras comunes y corrientes capturan una imagen bidimensional del mundo tridimensional en el que vivimos.

La esfera es quizá el cuerpo más fácil de intuir en cualquier sistema de coordenadas: es el conjunto de puntos que se encuentran a la misma distancia de un centro. Esto es de suma importancia porque si existen esferas en cualquier sistema de coordenadas, pueden inscribirse sólidos platónicos dentro de esas esferas y pueden obtenerse también curvas de Lissajou a partir de la parametrización de la superficie esférica (haciendo que la periodicidad en los parámetros tenga como mínimo común múltiplo a un número natural). Estas curvas de Lissajou en Rn tendrían también infinitas derivadas y serían ideales para trazar túneles y mover cámaras, trazando conos e intersectándolos con planos de n-1 dimensiones.

Partamos entonces de la parametrización de la esfera unitaria en R4

O[u,v,w] = { Cos[w], Cos[u] Sin[v] Sin[w], Sin[u] Sin[v] Sin[w], Cos[v] Sin[w] }

y de la curva de Lissajou

F[t,A,B,C] = O[t * A, t * B, t * C]

Con las tres primeras derivadas

F1[t,A,B,C] = D[F[t,A,B,C], t]

F2[t,A,B,C] = D[F1[t,A,B,C], t]

F3[t,A,B,C] = D[F2[t,A,B,C], t]

y el producto cruz de estas (producto cuña)

F4[t,A,B,C] = F1[t,A,B,C] x F2[t,A,B,C] x F3[t,A,B,C]

se obtienen 4 vectores linealmente independientes para cada punto de la curva F. Se puede volver a repetir el proceso, colocar cuerpos cada 2Pi / n segmentos a lo largo de t y realizar los cálculos necesarios para hacer las proyecciones a R3 y posteriormente a R2. La función para transformar cada punto P a un punto P' de coordenadas de vista desde la perspectiva de la cámara en F[t,A,B,C] es ahora la siguiente:

$$P' = (P - vPosicion) \cdot \begin{pmatrix} Xx & Yx & Zx & Nx \\ Xy & Yy & Zy & Ny \\ Xz & Yz & Zz & Nz \\ Xw & Yw & Zw & Nw \end{pmatrix}$$

donde

N = Unitario[F1[t,A,B,C]] (eje del cono)

X = Unitario[F2[t,A,B,C] x F3[t,A,B,C] x F1[t,A,B,C]] (nuevo eje X)

Y = Unitario[F3[t,A,B,C] x N x X] (nuevo eje Y)

Z = N x X x Y (nuevo eje Z)

vPosicion = F[t,A,B,C] (posición de la cámara, vértice del cono)

Luego es necesario intersectar los puntos P' con el plano de proyección para obtener los puntos P". Asumiendo que el plano de proyección está a una distancia igual a 1 del nuevo origen de coordenadas:

$$P'' = \frac{1}{Pw' * Tan[\theta/2]} \{Px', Py', Pz'\}$$

En las siguientes imágenes se colocaron 20 Cubos a lo largo de la curva F[t,3,2,2] y se movió la cámara a lo largo de F

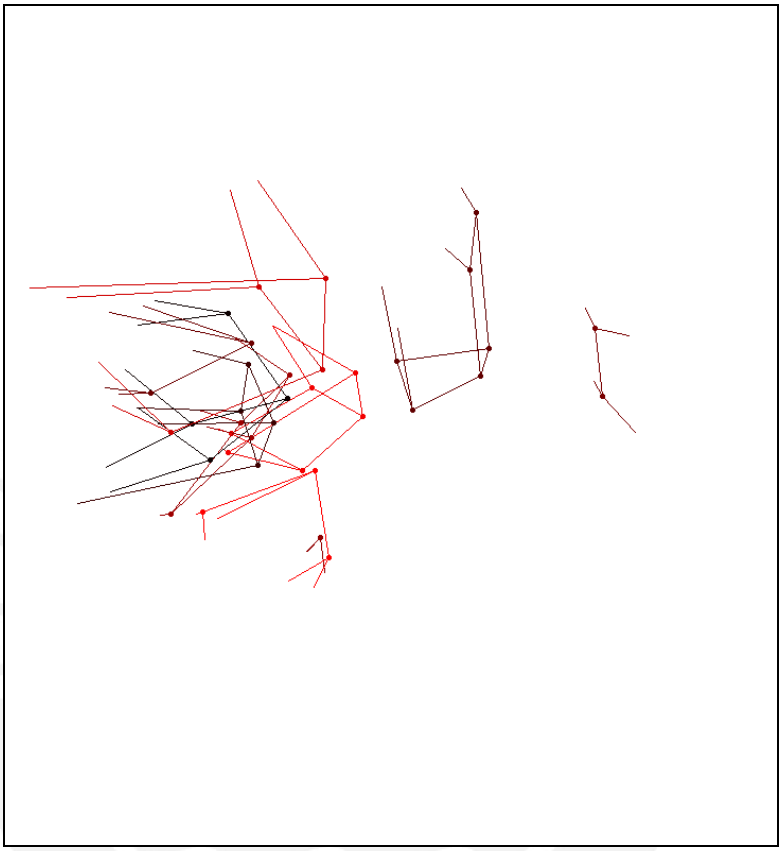


Figura 1.2.3-2

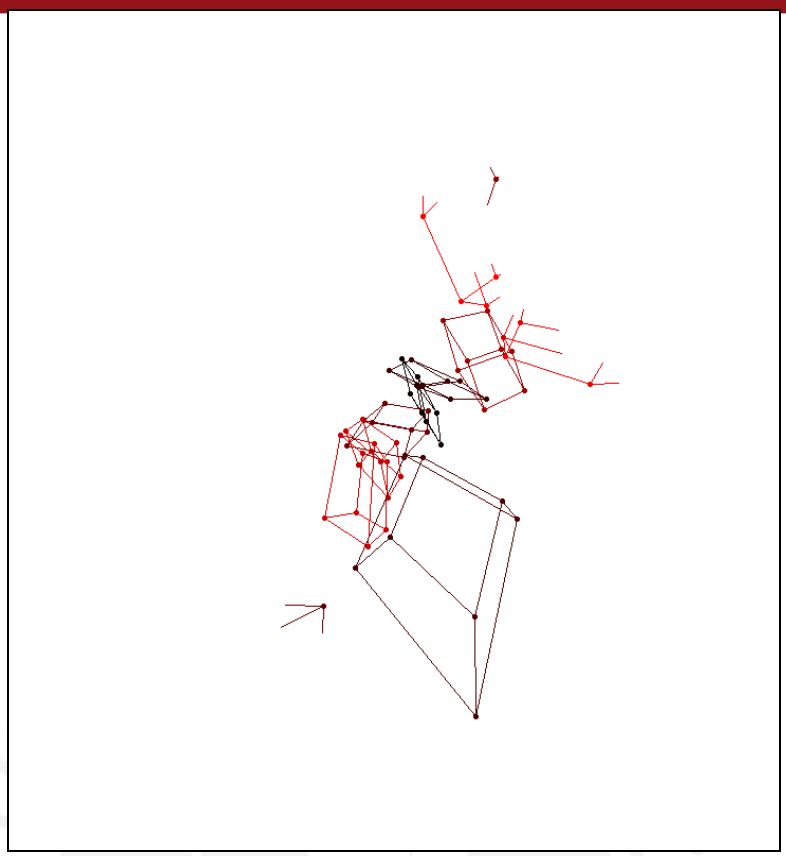


Figura 1.2.3-3

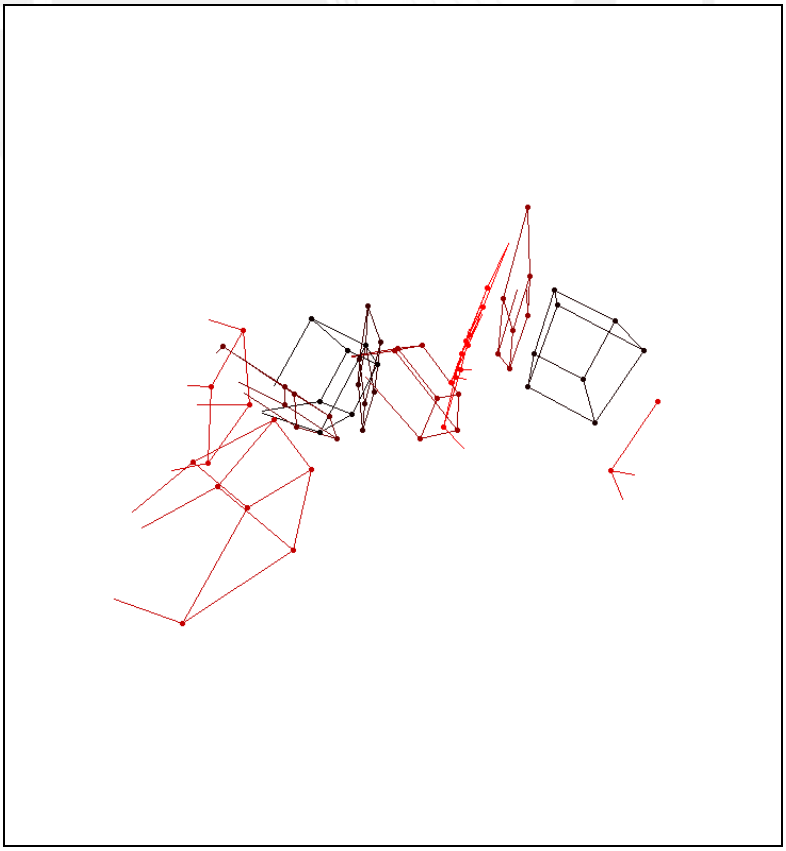


Figura 1.2.3-4

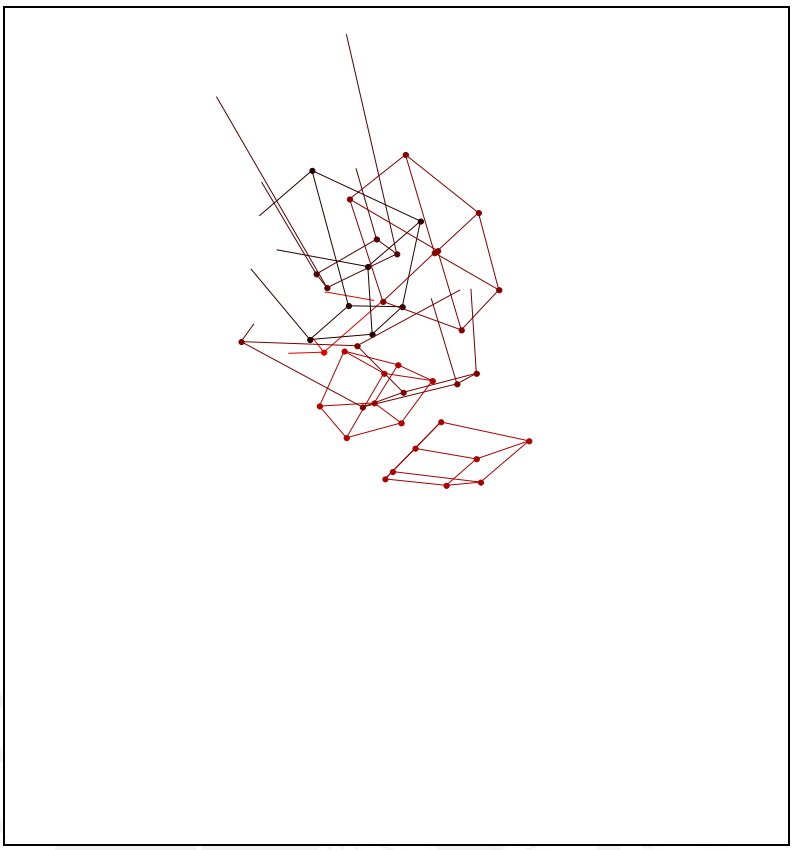


Figura 1.2.3-5

Nótese que estas imágenes fueron generadas con el software Mathematica 5.0 sin utilizar algoritmos para limitar rectas al rango visible (ver anexo).

Capitulo 2: Arquitectura

Este capítulo resume el paso de convertir la teoría y la información obtenida en el capítulo 1 en conceptos de diseño de software, como diagramas de clases y rutinas principales del programa.

2.1. Diagrama de clases de los elementos gráficos

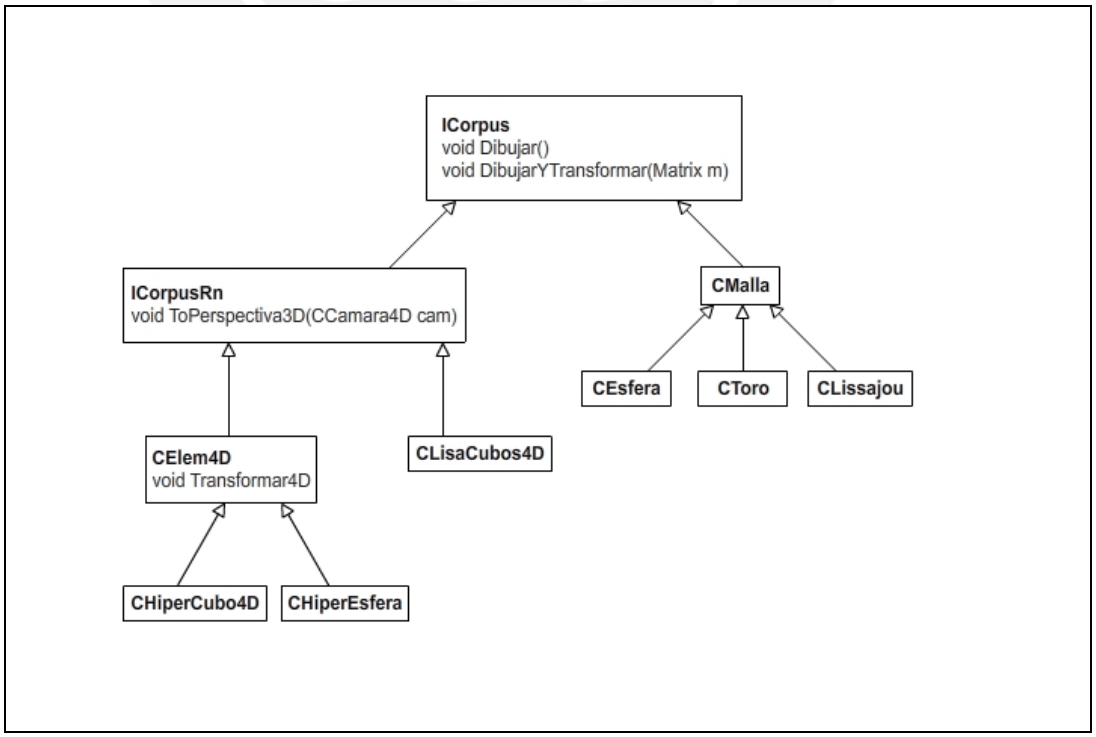


Figura 1.2.3-1

Dado que todos los elementos gráficos van a tener que dibujarse, es muy conveniente que todos tengan un ancestro común para aprovechar la herencia y el polimorfismo. Como se puede apreciar en la figura 2.1-1, la interfase ICorpus hace de ancestro común y obliga a todos sus hijos a implementar los métodos Dibujar y DibujarYTransformar.

La clase CMalla, por ejemplo, implementa esta interfase y permite crear grillas de puntos, por lo que las clases de elementos gráficos que pueden heredar de esta clase, son, precisamente, figuras geométricas que pueden crearse a partir de arreglos bidimensionales de puntos, como esferas, toros y demás superficies que pueden describirse con dos variables linealmente independientes.

Los elementos en R4 tendrán que heredar de una interfase ICorpusRn (hija de ICorpus) que además obligará a sus hijos a implementar el método ToPerspectiva3D. Este método es el paso más importante en la visualización de figuras geométricas de orden superior a 3 y en un análisis de posibles llamadas recursivas a un método similar estaría la clave para una generalización a Rn.

2.2. 'Pipeline' de la animación en tiempo real

El flujo principal del programa lo ejecuta un formulario, instancia de la clase FrmAnima (que hereda de Form), por medio de un bucle que dibuja y procesa eventos. La propiedad Anima (instancia de la clase CAnima) es la clase que administra los distintos elementos gráficos a dibujarse y que configura algunas rutinas necesarias para completar el dibujo de cada cuadro.

```
static void Main()
{
        Inicializar Ventana y Direct3D (se crea una instancia de CAnima)
        Mientras todo va bien con el formulario:

        while(frm.Created)
        {
                frm.Render();
                Application.DoEvents();
        }
}
```

En el método Render() de la clase CAnima es donde se da la secuencia de pasos para la animación:

```
private void Render()
{
        this.Anima.Escena_Inicio();

        this.Tempo();

        this.Anima.Dibujar();

        this.Anima.Escena_Fin();
}
```

Escena_Inico, entre otras cosas, limpia la zona de dibujo para empezar con un cuadro nuevo.

En el método Tempo() se llama a funciones de los distintos elementos de la animación que requieren parámetros de tiempo, como el movimiento de los objetos y la posición y orientación de la cámara.

Dibujar() llama al método Dibujar() de cada uno de los elementos gráficos. Aquí es donde aparecen todas las ventajas del diagrama utilizado ya que la herencia y el polimorfismo permiten que la clase CAnima guarde todos los objetos a Dibujarse en un único arreglo (un arreglo de objetos ICorpus) y los dibuje simplemente llamando a la clase Dibujar() de cada uno de los elementos del arreglo.

Luego, para proyectar a R3 figuras geométricas en R4 y dibujarlas, es necesario especificar una secuencia de pasos que irán ejecutándose a lo largo del bucle principal del formulario FrmAnima.

Secuencia 1: Proyección a R3 de figuras geométricas en R4:

1) Ubicar los puntos en el espacio de cuatro dimensiones.
2) Trazar las rectas (aristas) entre los distintos puntos de la figura geométrica y guardarlas en un arreglo.
3) Para cada recta del arreglo, evaluar si ambos extremos están contenidos dentro del cono que determina el rango visible y restringir su tamaño al segmento de recta contenido dentro del cono (ver apéndice).
4) Proyectar cada recta sobre el hiperplano (el plano de proyección) que tiene como normal un vector paralelo al eje del cono.
5) Dibujar el arreglo de rectas en 3D.

La Secuencia 1, sin embargo, refleja la idea de los pasos necesarios para llevar a cabo la proyección de R4 a R3 pero no considera cómo esos pasos deberían darse en una arquitectura orientada a objetos y considerando el bucle principal anteriormente descrito. Por lo tanto, a partir de la Secuencia 1, definiremos la Secuencia 2 para hacer el proceso mucho más eficiente.

Secuencia 2: Proyección a R3 de figuras geométricas en R4 orientada a objetos:

1) Realizar los pasos 1 y 2 de la Secuencia 1 en el constructor o en un método de la clase y ubicar los vértices de la figura geométrica en R4 centrada en el origen de coordenadas (ejecutar este paso antes de iniciar el bucle principal para no generar un desbordamiento de la memoria).
2) Llamar al método 'Transformar4D' para trasladar y transformar los puntos en cada cuadro de la animación (en el método 'Tempo' de 'FrmAnima').
3) Llamar al método 'ToPerspectiva3D' para proyectar los puntos al hiperplano (pasos 3 y 4 de la Secuencia 1, también dentro del método 'Tempo').
4) Llamar al método 'Dibujar'.

En el bucle principal, al llamarse al método 'Render' de 'FrmAnima', ocurriría lo siguiente:

// Se borra la ventana principal con el último cuadro de la animación
this.Anima.Escena_Inicio();

// Se transladan y transforman todos los objetos, incluida la cámara que no es más
// que un cono en R4, utilizando una variable t que se incrementa en cada cuadro

```
// Se llama al método 'ToPerspectiva3D' de todos los objetos a dibujarse
this.Tempo();

// Se dibujan todos los objetos
this.Anima.Dibujar();

// Se envían eventos para informar que el cuadro esta listo y ha sido dibujado
this.Anima.Escena_Fin();
```

Capitulo 3: Extendiendo algunos conceptos a Rn

Tomando los conceptos del capítulo 1 y la arquitectura del capítulo 2, el presente capítulo intenta llegar a una generalización a Rn del concepto de hipercubo, como ejemplo de una figura geométrica fácil de construir y que puede ser dibujada en todas las dimensiones por encima de R2. Contando con una figura geométrica de estas características y considerando las demás figuras descritas en el capítulo 1, es posible intuir una generalización a Rn de la proyección cónica tanto a nivel geométrico, como de implementación.

3.1. El Hipercubo

Como se menciónó en el capítulo 1.3, figuras geométricas como el cono, la esfera y el plano pueden ser generalizadas a Rn dadas las ecuaciones que las describen. El cubo puede extenderse también a Rn dado que tiene la propiedad de que sus aristas son linealmente independientes entre sí y pueden ser siempre alineadas con cada una de las componentes del sistema.

En el hipercubo, para cada uno de sus puntos se cumplirá que, en cada vector que define al sistema de coordenadas, habrá únicamente dos valores posibles. Esto equivale a un sistema binario en el que cada bit representa una dimensión con dos valores posibles.

Diremos entonces que un cubo en Rn, o un hipercubo en Rn:

- tendrá siempre 2^n vértices (siendo n el número de dimensiones del sistema)
- que las coordenadas de dichos vértices podrán ser halladas obteniendo el valor binario del número de vértice.

Para un cuadrado, por ejemplo, n es igual a 2 (el cuadrado existe en R2), por lo tanto 2^2 vértices es igual a 4. Luego, a partir del número de cada vértice en binario, se obtiene:

V0 = 0 = 00 = {0,0} = coordenadas del primer vértice
V1 = 1 = 01 = {0,1} = coordenadas del segundo vértice
V2 = 2 = 10 = {1,0} = coordenadas del tercer vértice
V3 = 3 = 11 = {1,1} = coordenadas del cuarto vértice

lo que corresponde a los vértices de un cuadrado de arista igual a 1 con uno de sus vértices en el origen. Para un hipercubo en R4, la cantidad de vértices sería 2^4 = 16 y las coordenadas de los vértices:

V0 = 0 = 0000 = {0,0,0,0}
V1 = 1 = 0001 = {0,0,0,1}
V2 = 2 = 0010 = {0,0,1,0}
V3 = 3 = 0011 = {0,0,1,1}
V4 = 4 = 0100 = {0,1,0,0}
V5 = 5 = 0101 = {0,1,0,1}
V6 = 6 = 0110 = {0,1,1,0}
V7 = 7 = 0111 = {0,1,1,1}
V8 = 8 = 1000 = {1,0,0,0}
V9 = 9 = 1001 = {1,0,0,1}
V10 = 10 = 1010 = {1,0,1,0}
V11 = 11 = 1011 = {1,0,1,1}
V12 = 12 = 1100 = {1,1,0,0}
V13 = 13 = 1101 = {1,1,0,1}
V14 = 14 = 1110 = {1,1,1,0}
V15 = 15 = 1111 = {1,1,1,1}

Luego, para trazar las aristas bastaría con dos sentencias For enlazadas para comparar todos los puntos entre sí y dibujar líneas sólamente si la distancia entre ambos puntos es igual al tamaño de la arista.

Entonces, dado el arreglo de vértices arrVertices, bastaría con ejecutar:

```
For( int i = 0; i < arrVertices.Length-1; i++)
  For( int j = i + 1; j < arrVertices.Length; j++)
    If( distancia( arrVertices[i], arrVertices[j] ) == long_arista )
      dibujarLinea( arrVertice[i], arrVertice[j] );
```

Con esta generalización a Rn, es posible dibujar hipercubos en sistemas de coordenadas superiores a 3, sin embargo, lo complicado sería hacer un algoritmo que vaya proyectando la figura recursivamente a planos de orden n-1 hasta llegar a R3.

Cabe mencionar que el algoritmo anteriormente descrito siempre calcula los puntos del hipercubo con un vértice (el vértice 0) en el origen de coordenadas del sistema. Para mover y centrar al hipercubo en el origen, basta con remplazar los ceros por -1 en las coordenadas obtenidas a partir del número de vértice en binario. Haciendo el reemplazo en el ejemplo del cuadrado, la nueva posición de los vértices sería:

V0 = 0 = 00 = {0,0} = {-1,-1}
V1 = 1 = 01 = {0,1} = {-1,1}
V2 = 2 = 10 = {1,0} = {1,-1}
V3 = 3 = 11 = {1,1} = {1,1}

Lo que da un cuadrado centrado en el origen de coordenadas. Ahora sí se puede tomar como ejemplo la clase CHiperCubo4D siguiendo la Secuencia 2 descrita en el capítulo 2.2:

1) Antes del inicio del bucle principal, el constructor de la clase generará los vértices de un hipercubo centrado en el origen de coordenadas y de arista igual a uno.
2) Dentro de Tempo, el hipercubo podrá ser trasladado y transformado cuadro por cuadro. Luego, tomando en cuenta la posición y dirección de la cámara, cada una de las aristas será limitada al rango visible y proyectada sobre el plano de proyección.
3) Finalmente, llamando al método Dibujar se dibujarán las rectas proyectadas.

3.2. Proyección recursiva de Rn hasta R2

Para proyectar un conjunto de puntos desde Rn hasta un plano en R2 es necesario repetir recursivamente el procedimiento de generar la proyección cónica de los puntos en Rn a un hiperplano de n-1 dimensiones y de normal paralela al eje del cono.

El principal problema que surge entonces es que tanto la 'cámara' como el objeto a ser proyectado existen en un universo superior a la imagen proyectada. Una vez lograda la proyección la nueva cámara (el nuevo cono de N-1 dimensiones) no tendrá ninguna relación con la cámara que generó el universo donde la nueva cámara existe. Por lo tanto, resultará imposible determinar en qué posición del hiperplano de proyección debe ubicarse. Si bien es cierto que, intuitivamente, se puede ubicar a la nueva cámara en el origen de coordenadas (en el centro de la 'imagen' resultante), el objetivo de la nueva cámara podría apuntar hacia cualquier lado del nuevo espacio de n-1 dimensiones.

Por lo tanto, salvo que se trabaje en un algoritmo que fije arbitrariamente la posición y orientación de las nuevas cámaras, la idea de proyectar puntos desde Rn hasta R2 implica n-2 cámaras flotando a la deriva por dimensiones distintas y algunas veces, con los puntos a proyectarse fuera del rango visible. Sin embargo, lo interesante de este proceso sería que la imagen final (es decir, la proyección recursiva del arreglo de cámaras para un instante de tiempo) no sería tan relevante para conocer el objeto como el movimiento de las cámaras en el tiempo alrededor de las sucesivas proyecciones.

Una clase que administre este arreglo de cámaras con parámetros globales y que las oriente hacia los puntos de proyección sería una interesante extensión de este trabajo, así como la búsqueda de una especie de sólido platónico, capaz de permanecer inmutable en cada una de las proyecciones desde Rn hasta R2.

Capitulo 4: Observaciones y conclusiones

Principales observaciones y conclusiones sobre el desarrollo de la tesis y la implementación final del programa.

4.1. Observaciones

Observacion 1: Este trabajo de tesis no cuenta con un capítulo que trate sobre el estado del arte del tema ya que no he podido encontrar productos de visualización de datos en más de 3 dimensiones para usuarios finales. Diría que el mercado está enfocado principalmente en el lucrativo negocio del entretenimiento con juegos de video y películas de animación en 3D. Herramientas como 3DStudio, Cinema4D y Maya son algunas de las herramientas para hacer lo que se conoce como modelamiento y animación en 3D.

Si bien es cierto que existen varias imágenes en la red sobre politopos en el hiperespacio y programas que grafican quaterniones y fractales en 4 dimensiones, me parece que el no trabajar con conceptos simples de figuras geométricas que conocemos todos, restringe este tipo de herramientas a gente con un entendimiento mucho más profundo de geometría y matemáticas. De ahí la necesidad y la apuesta de este trabajo por extender conceptos de figuras simples como esferas, cubos, planos, conos, etc, a generalizaciones en Rn.

Es precisamente al extender estos conceptos y al tomar a una de estas figuras simples (el cono) como principal herramienta de proyección que es posible y tiene sentido la idea de proyectar cuerpos desde Rn hasta el plano bidimensional.

Observación 2: En R4, surge un efecto como de ojo de pez, en el que partes del objeto que deberían estar juntas se pierden hacia extremos opuestos de la proyección. En la figura 4.1.1 se puede apreciar como lo que debería ser una curva continua cuyos colores van del roja al blanco, parece abrirse en direcciones distintas.

Figura 1.2.3-1

Observación 3: Para la rotación de cuerpos en el espacio de más de tres dimensiones se vuelve más relevante la rotación con respecto a dos o más ejes a la misma vez. Desarrollar algoritmos de rotación para girar objetos con respecto a un plano o a un espacio tridimensional sería un aporte interesante.

De la misma manera, el control de varias variables a través de parametrizaciones con una o pocas variables sería algo muy conveniente para mover y transformar cuerpos en el espacio, como se demostró con las curvas de Lissajou en el primer capítulo.

4.2. Recomendaciones finales y posibles extensiones

A continuación una lista de recomendaciones y posibles extensiones que me hubiese gustado llevar a cabo y que, pienso, serían temas muy interesantes de investigación:

1. Orientar el control y creación de figuras geométricas a un lenguaje matemáticamente más formal: una herramienta que permita la creación de politopos multidimensionales con la nomenclatura de Schläfli como input sería una herramienta bastante ilustrativa.

2. Algo que me llamó la atención al ver imágenes de politopos en internet fue que siempre se utilizaba una proyección paralela para dibujarlos, quizá para resaltar el alto grado de simetría de las figuras. Lo cierto es que este tipo de proyección no es compatible con el concepto de cámara, que implica una proyección cónica, un foco, donde convergen las líneas de proyección y se

ubica el observador. Una propuesta para lograr visualizar animaciones utilizando una proyección paralela sería algo que valdría la pena investigar.

3. Dado que de un mundo en R4 resulta una proyección tridimensional, el no tener cómo ubicar y orientar a la nueva cámara 3D en el nuevo espacio tridimensional proyectado, significa un verdadero problema. Recordemos que, una vez en R3, aun será necesario hacer una proyección de R3 a R2 para lograr la vista bidimensional final que aparecerá en pantalla. Por lo tanto, la ubicación arbitraria de la cámara plantea un problema cuya solución sería clave para lograr una generalización a Rn del proyecto y una forma de proyectar sucesivamente de Rn a Rn-1 hasta llegar R2.

4. Por último, me hubiese gustado programar un visualizador de gráficos en R4 en un entorno de varias pantallas en red. Quizá, para alguien con los conocimientos suficientes en sockets y sincronización de eventos o herramientas como .NET Remoting tal extensión del trabajo podría resultar interesante. Sería un excelente tema a desarrollar para un grupo de investigación interdisciplinario que incluya gente de áreas como matemáticas, informática y telecomunicaciones. El 2006 propuse el tema a la DAI, pero, para mi enorme decepción, me respondieron que las políticas de la universidad, al menos por aquel entonces, impedían que un alumno estuviese a cargo de una investigación.

4.3. Conclusiones

Ha sido muy positivo y enriquecedor el aterrizar ideas abstractas en diagramas de clases y demás documentos de diseño de software. Si bien es cierto que, en la mayoría de los casos, mi experiencia en este campo se limitaba a casos de la vida cotidiana como procesos al interior de una empresa (venta de productos, administración de recursos, etc), el tener que pensar en términos de análisis y diseño de software conceptos de números, geometría y gráficos, me ha parecido hasta ahora la aplicación más interesante de estos conocimientos y espero poder encontrar un campo de desarrollo para seguir aprendiendo más sobre este híbrido entre matemáticas y programación.

Bibliografía

Para leer sobre matrices de transformación y traslación:

Gráficas por Computadora de Donald Hearn y M. Paulin Backer, Prentice Hall
Calculus and Analytic Geometry de George B. Thomas Jr, Departament of Mathematics – Massachussets Institute of Technology

Como ayuda para programar pruebas en un entorno simple orientado a las matematicas:

The Mathematica Book de Stephen Wolfram, Wolram Media

Anexos

Limitación de rectas al rango visible

Considerando al rango visible como un cono, existen cuatro posibles casos en los que un segmento de recta y un cono pueden ubicarse en el espacio:

a) **Caso 1:** que la recta esté completamente dentro del rango visible

   Este es el caso más simple, donde ambos puntos de la recta pueden ser transformados a puntos proyectados sobre un espacio de n-1 dimensiones (recordar que una de las dimensiones se perderá al ser usada como la normal del hiperplano de proyección) para luego dibujar la recta que los une.

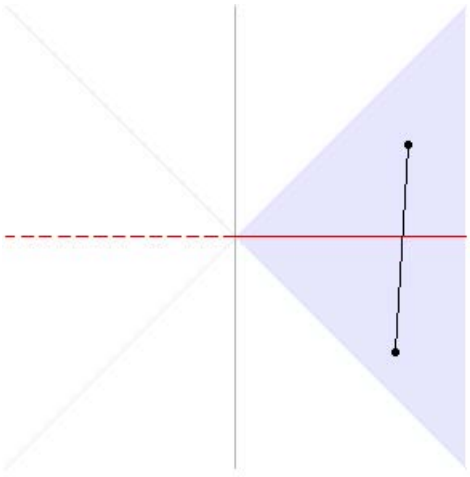


Figura caso 1. 0-1

Es necesario aclarar que el rango visible (en la imagen 1.1, de color gris) es solamente una mitad de lo que matemáticamente podría considerarse como 'rango visible' ya que la solución de la ecuación del cono, planteada vectorialmente, contiene además una región igual a la sombreada (con el vértice del cono como punto de simetría) al lado izquierdo de la imagen.

Por lo tanto, es necesaria la restricción de que el vector que une el vértice del cono con cualquier punto en el rango visible, debe tener una componente positiva sobre el vector 'eje' (línea continua en rojo).

b) **Caso 2:** que un extremo de la recta quede fuera del rango visible

En la figura 2.1 se puede apreciar fácilmente que el vector trazado desde el vértice del cono hasta el punto que cae fuera del rango visible, tendría una proyección negativa sobre el vector 'eje' y, por lo tanto, debería quedar excluido inmediatamente del conjunto de puntos dentro del rango visible, por más que, de continuar la recta hacia la izquierda, el punto volvería a caer dentro de lo que matemáticamente se consideraría el interior del cono.

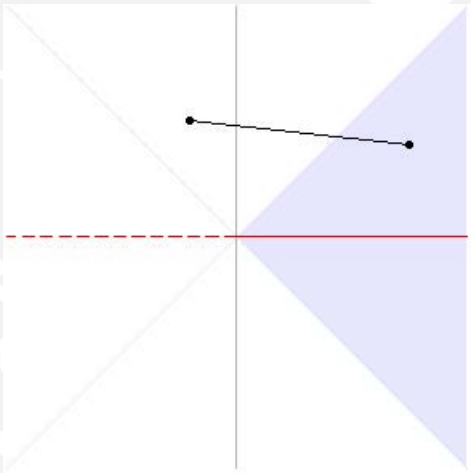


Figura caso 2. 1

Sin embargo, en este caso, es necesario restringir la recta al segmento dentro del rango visible y reemplazar al punto fuera del cono por el punto que determina la intersección de la recta con la superficie del cono. Entonces, la recta a proyectarse sería la que en la imagen 2.2 se muestra como una línea continua negra determinada por dos puntos dentro del cono, uno de ellos exactamente en el límite del rango visible.

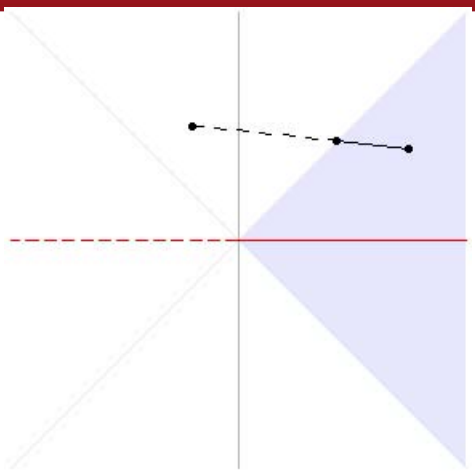


Figura caso 2. 2

Para hallar las coordenadas de este punto en el que la recta sale del rango visible, es necesario plantear el problema como una suma vectorial en la que, como se aprecia en la imagen 2.3, la suma de vectores grises es igual al vector azul, vector cuyo vector unitario multiplicado por el eje del cono será igual al ángulo de apertura del cono (en este caso Pi/4).

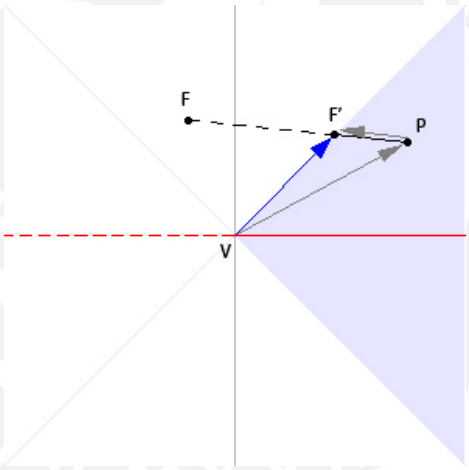


Figura caso 2. 3

Para el punto de intersección F', es válida la siguiente igualdad:

v_azul = (F'-V) = (P-V) + (F'-P)

Luego, utilizando una variable t para escribir F' en función de P y F:

v_azul = (P-V) + (F'-P) = (P-V) + (F-P) * t

Como el vector v_azul y el eje del cono forman siempre un ángulo igual al ángulo de abertura del cono:

v_azul . eje = ||v_azul|| * ||eje|| * cos[angulo]

Y dado que el vector eje es un vector unitario la ecuación se simplifica a:

v_azul . eje = ||v_azul|| * cos[angulo]

Reemplazando:

((P-V) + (F-P) * t) . eje = ||(P-V) + (F-P) * t|| * cos[angulo]

con t como única incógnita. Despejando se obtiene dos valores: uno en el rango de [0,1] que es la solución correcta para obtener F', y otro, fuera del rango de [0,1], que vendría a ser el punto en el que la prolongación de la recta PF corta a la recta y=-x, entrando en el cono de la izquierda, también considerado matemáticamente como parte de la solución pero no como parte del rango visible.

Las coordenadas del punto de intersección se obtienen calculando:

P + (F-P) * t

Donde se aprecia más claramente por qué t tiene que estar en el rango de [0,1] y por qué a simple vista la segunda solución es mayor que 1 ya que evaluando P + (F-P) * t para t = 1 se obtiene F y para t = 0, P.

c) **Caso 3:** que la recta este completamente fuera del rango visible

Este podría parecer el caso más simple de todos, porque entonces se tendría que excluir a la recta del proceso de proyección y transformación.

Sin embargo, como la recta se conoce simplemente por los dos puntos que la delimitan, podría confundirse el que ambos puntos caigan fuera del rango visible, con que la recta este efectivamente fuera del rango visible, como se muestra en la imagen 3.1.

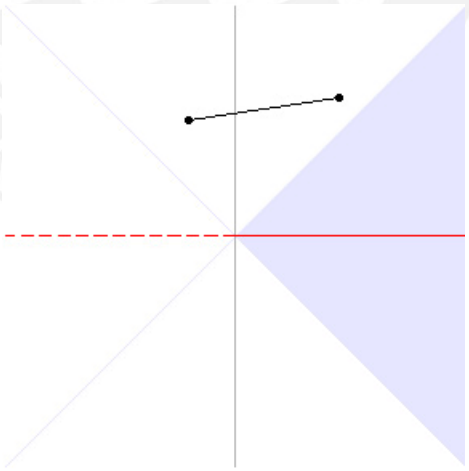


Figura caso 3. 1

Por lo tanto, es necesario considerar un cuarto caso en el que ambos puntos estén fuera del rango visible pero que algún segmento de la recta lo atraviese.

d) **Caso 4:** que ambos extremos queden fuera del rango visible pero que algún segmento de la recta lo cruce

Este es el caso que se muestra en la imagen 4.1, con ambos puntos fuera del cono pero con una parte de la recta dentro del rango visible.

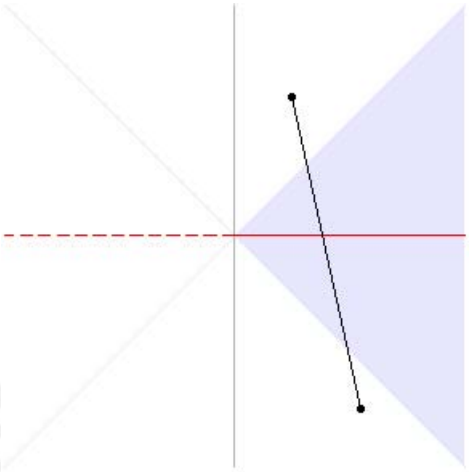


Figura caso 4. 1

En este caso será necesario, al igual que en el caso 2, limitar la recta al segmento contenido dentro del rango visible.

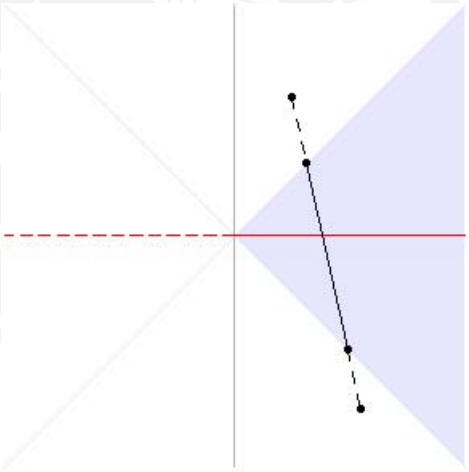


Figura caso 4. 2

La ecuación para encontrar ambos puntos es la misma que la del caso 2 sólo que, en este caso, ambas soluciones de la ecuación se encontrarán en el rango de [0,1] y determinarán los puntos del nuevo segmento de recta contenido dentro del rango visible.