***Títol: Optimització dels valors d'opacitat com a gestió de l'oclusió 3D***

***Volum: 1 de 1***

***Alumne: Joan Serrat Alarcón***

***Director/Ponent: Carlos Antonio Andújar Gran***

***Departament: LSI***

***Data: Juny de 2010***

**DADES DEL PROJECTE**

*Títol del Projecte: Optimització dels valors d'opacitat com a gestió de l'oclusió 3D*

*Nom de l'estudiant: Joan Serrat Alarcón*
*Titulació: Enginyeria Informàtica*
*Crèdits: 37,5*
*Director/Ponent: Carlos Antonio Andújar Gran*
*Departament: LSI*

---

**MEMBRES DEL TRIBUNAL** *(nom i signatura)*

*President: Isabel Navazo Alvaro*

*Vocal: Miquel Noguera Batlle*

*Secretari: Carlos Antonio Andújar Gran*

---

**QUALIFICACIÓ**

*Qualificació numèrica:*
*Qualificació descriptiva:*

*Data:*

---

# ÍNDEX DE CONTINGUTS

# 1 INTRODUCCIÓ

## 1.1 DESCRIPCIÓ

En el camp dels gràfics per computador, les escenes 3D amb una gran quantitat d'objectes cada cop són més habituals. Escenes com els interiors d'edificis tenen un alt nivell d'oclusió, que fa que donat un determinat punt de vista només sigui visible una petita fracció dels objectes de l'escena (figura 1-1).

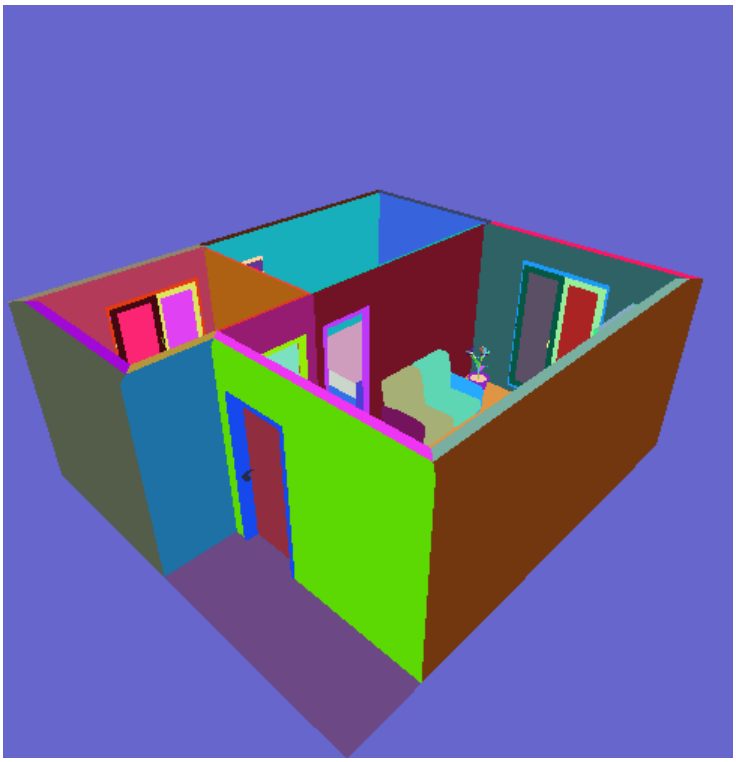


**Figura 1-1. Degut a l'oclusió alguns objectes de l'escena no són visibles des d'un determinat punt de vista**

A la literatura especialitzada s'han proposat diferents tècniques per reduir els problemes derivats de l'oclusió dels objectes, i permetre que l'usuari pugui veure objectes altrament oclusos. Una tècnica molt simple consisteix en dibuixar els objectes amb semi transparència (*alpha blending*), per evitar que els objectes més propers tapin totalment els que tenen darrere (figura 1-2). Aquesta tècnica requereix definir, a nivell d'objecte o de primitiva, un valor d'opacitat, anomenat *alpha*. Una possibilitat consisteix en assignar una opacitat uniforme a tots els objectes (per exemple 50%); aquesta opció,

però, només dona resultats acceptables en escenes de baixa complexitat. Una altra opció consisteix en assignar els valors d'opacitat manualment, però en escenes amb un gran nombre d'objectes aquesta tasca requereix un esforç considerable que a més depèn del punt de vista triat.

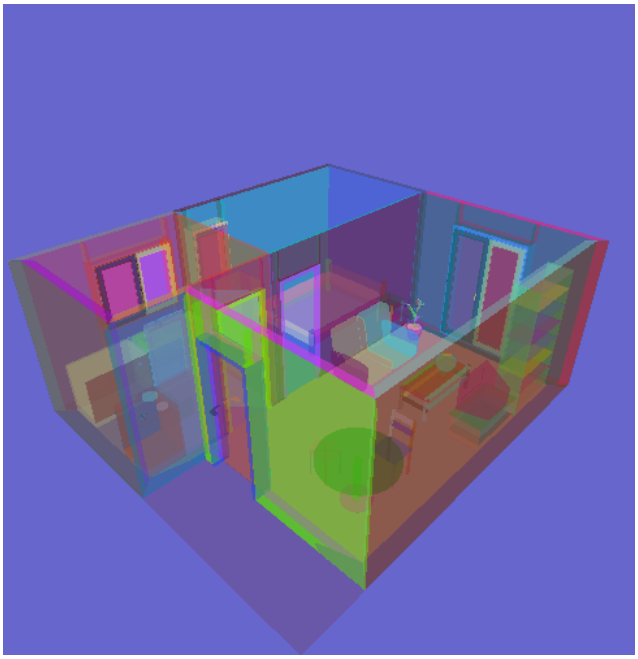


**Figura 1-2. Imatge amb objectes semi transparents.**

## 1.2 OBJECTIUS

Els objectius que s'han d'assolir en aquest projecte són els següents:

1) Dissenyar i implementar un heurístic capaç de determinar donades unes imatges generades amb transparències, quina és la que permet a l'ull humà distingir un nombre major d'objectes.
2) Donat aquest heurístic, optimitzar els valors d'opacitat de cada un dels objectes del model per tal de maximitzar el nombre d'objectes reconeguts per un usuari.
3) Estudiar i implementar diferents mètodes d'assignació de valors d'opacitat als objectes d'una escena.
4) Realitzar tests amb usuaris per tal d'establir quines solucions són més adients segons les tasques a realitzar i comparar els resultats amb els obtinguts mitjançant l'heurístic.

# 2 TREBALL PREVI

## 2.1 INTRODUCCIÓ

Les tècniques utilitzades per solucionar els problemes d'escenes amb oclusió es poden classificar en cinc grups tal i com descriuen N. Elmqvist i P.Tsigas (2008), que enfoquen el problema de diferents maneres.

- Múltiples *viewports*: El conjunt de tècniques agrupades en aquest apartat consisteixen en mostrar la mateixa escena des de varis punts de vista, aconseguint d'aquesta manera veure parts que d'altra manera estarien tapats (figura 2-1).

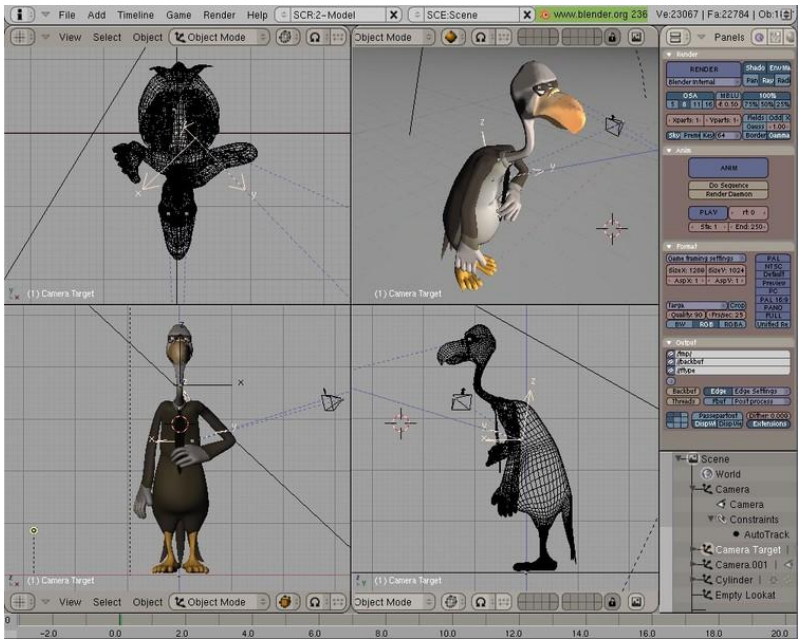



**Figura 2-1. Gestió de l'oclusió amb múltiples viewports**

- Raigs X virtuals: Aquestes tècniques permeten fer semi transparents parts dels objectes de la escena per mostrar els objectes que d'altra manera no seria possible veure. Les tècniques poden ser actives, on l'usuari escull les zones que es fan semi transparents, o passives, en la que l'aplicació automàticament mostra els objectes.
- *Tour planner*: La família de tècniques de *tour planner* funcionen de la mateixa manera. Primer de tot, hi ha una fase de planificació, on es

calcula una ruta que passa per diferents zones de l'escena i a continuació, es "viatja" pel camí de manera interactiva.

- *Volumetric probe*: Les tècniques agrupades en aquest apartat modifiquen la posició dels objectes utilitzant alguna forma tridimensional. Per exemple, la tècnica de *BallonProbe*, en N. Elmqvist, 2005, s'utilitza una esfera, que es situa en una posició de l'escena, s'infla o es desinfla alterant la posició de la resta d'objectes de l'escena (figura 2-2).

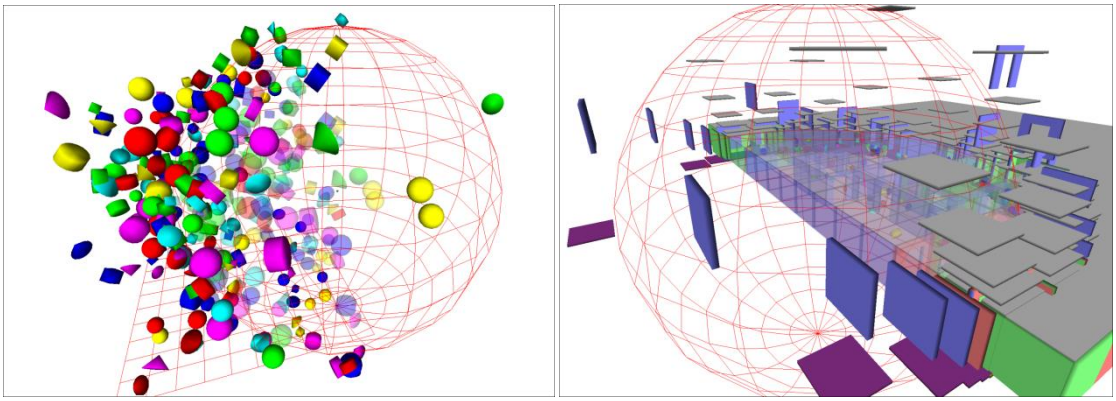


**Figura 2-2. Gestió de l'oclusió amb *volumetric probes*.**

- *Distorsió de projecció*: Aprofitant les característiques de la vista projectiva i paral·lela i alternant-les (o generant una animació que canvia d'una a l'altra) permet adquirir informació que d'altra manera seria molt complicat d'utilitzar (figura 2-3).

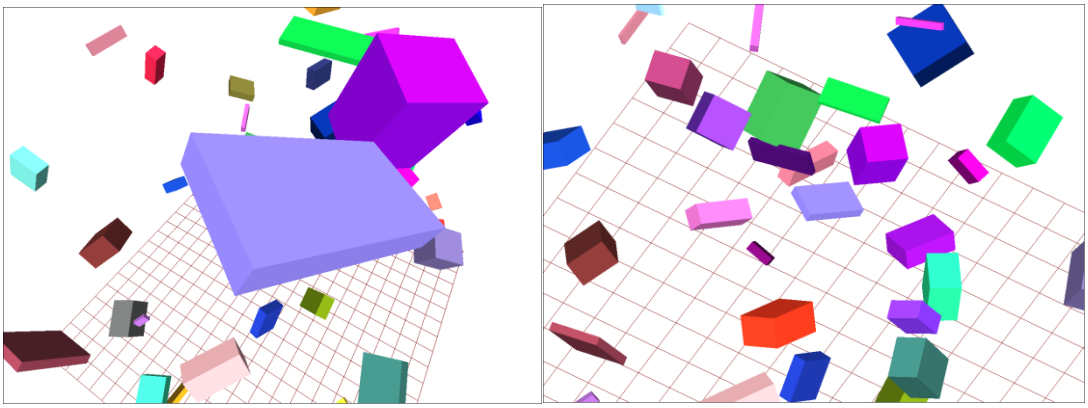


**Figura 2-3. Vistes perspectiva i paral·lela de la mateixa escena.**

Les tècniques que s'exploren en aquest projecte, que són les que treballen amb objectes semi transparents, formen part de les tècniques de raigs X virtuals.

## 2.2 TÈCNIQUES DE RAIGS X VIRTUALS

El mètode més utilitzat per a tractar amb objectes semi transparents és la tècnica anomenada *alpha blending*, que consisteix en donar una opacitat a cada objecte i composar el color de sortida ponderant el color ja existent amb el color del nou objecte utilitzant les transparències dels objectes.

Per tal de composar el color en l'ordre correcte es pot obtenir a priori un conjunt d'imatges amb la tècnica de *depth peeling*. L'adquisició del conjunt de les imatges s'explicarà posteriorment a l'apartat 4.1, però en aquests moments només cal saber que aquestes imatges representen varies capes del model. La primera té els elements més propers a l'observador, la segona els elements que estan tapats pels elements que apareixen en la primera imatge i així successivament.

Un cop obtingut el conjunt de les imatges es procedeix a fer una composició de colors, on per a cada píxel de la imatge resultant, es calcula el seu color seguint la següent fórmula, que en aquest exemple té només 2 capes més el color de fons:

$$C = C_1 \cdot \alpha_1 + (1 - \alpha_1) \cdot (C_2 \cdot \alpha_2 + (1 - \alpha_2) \cdot F)$$

Aquest mètode de composició de color està suportat nativament en *OpenGL* mitjançant la funció de *glBlendFunc* la qual cosa ens permet obtenir els resultats d'aplicar aquesta tècnica en temps real.

La funció *glBlendFunc* calcula el color que es mostra en pantalla utilitzant el color de l'objecte que s'està pintant actualment i el color que ja hi ha al *buffer* de color, assignant una ponderació a cada color seguint els paràmetres introduïts en la funció.

Cal tenir en compte que si es vol utilitzar *OpenGL* per calcular el color, els objectes s'han de pintar en ordre, de més llunyà a més proper. En el cas de no realitzar aquestes accions, el color resultant no és correcte.

S'han avaluat altres tècniques que treballen amb transparències, essent la més interessant la que es tracta en l'article de N. Elmqvist, U. Assarsson i P. Tsigas (2007) i en el de U. Assarsson, N. Elmqvist i P. Tsigas (2006). Els autors proposen agrupar els objectes de l'escena en 2 grups: objectes objectius, que són els que sempre es volen veure, i objectes que tapen. Un cop estan els objectes agrupats, s'implementa l'algorisme seguint un conjunt de regles:

1. Tots els objectes objectius han de ser visibles des de qualsevol posició.
2. Un objecte tapat es fa visible modificant l'opacitat dels píxels de la superfície que el tapa.
3. Les superfícies es poden fer impenetrables i mai poden ser convertides en transparent.
4. Els objectes poden tapar-se a sí mateixos.

Després de definir les regles la implementació del mètode és ben senzilla:

1. Els objectes es pinten de més llunyans a més propers.
2. Si l'objecte actual és un objecte objectiu, es modifica una màscara *alpha*. Altrament, es combina el color de l'objecte tenint en compte la màscara *alpha*.

La màscara *alpha* consisteix en una matriu de la mida del *viewport* que emmagatzema la opacitat del píxel corresponent. A l'aplicar l'algorisme, s'aconsegueix que els objectes objectius sempre siguin visibles, ja que els píxels corresponents a les superfícies que els tapen tenen una opacitat inferior a 1. Per aconseguir aquest efecte, en el moment de pintar els objectes objectius es modifiquen els píxels corresponents de la màscara *alpha*. El resultat es pot veure en la figura 2-4.

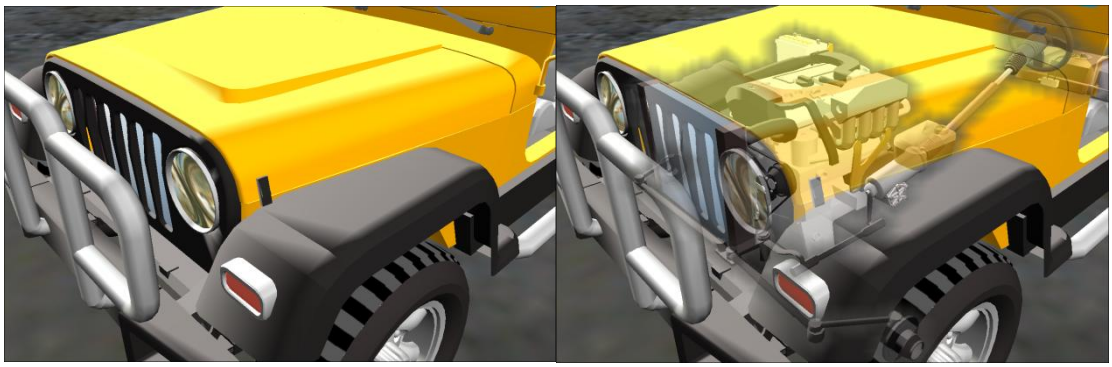


**Figura 2-4. Gestió de l'oclusió segons N. Elmqvist, U. Assarsson i P. Tsigas (2007)**

Tot i semblar resultar ser una bona tècnica comporta vàries diferències amb els objectius d'aquest projecte. El més evident de tots és que aquesta solució busca maximitzar la visualització d'uns objectes en concret enlloc de maximitzar el nombre d'objectes que es poden veure en una escena. Per altra banda, en aquesta tècnica cal un procés previ en el que es decideix quins són els objectes objectius i quins són els que tapen, donant importància només a un petit grup d'objectes.

# 3 DESENVOLUPAMENT TÈCNIC

## 3.1 DISSENY I IMPLEMENTACIÓ DE L'APLICACIÓ

### 3.1.1 ANÀLISI DE REQUISITS

L'anàlisi de requisits té com a objectiu determinar quines funcionalitats ha de complir l'aplicació. Generalment, aquestes funcionalitats s'obtenen a partir d'entrevistes als usuaris finals de l'aplicació i a les persones que sol·liciten l'aplicació. Els requisits es poden dividir en dues categories:

- Requisits funcionals: Indiquen les funcionalitats que té l'aplicació, com per exemple, que l'usuari ha de poder carregar un model.
- Requisits no funcionals: Són el conjunt de condicions i restriccions de l'aplicació. Per exemple, l'aplicació s'ha de poder executar en Linux.

#### 3.1.1.1 REQUISITS FUNCIONALS

Els requisits funcionals de l'aplicació ha de complir són els següents:

- L'aplicació ha de ser capaç de carregar models en format OBJ.
- L'aplicació ha de mostrar en una finestra el model seleccionat i ha de ser capaç de permetre a l'usuari modificar el punt de vista, el zoom, etc.
- L'aplicació ha de poder generar un conjunt d'imatges que seran el resultat d'aplicar les diferents tècniques proposades i emmagatzemar-les en format BMP.

#### 3.1.1.2 REQUISITS NO FUNCIONALS

Els requisits no funcionals es mostren a continuació:

- L'aplicació estarà preparada per funcionar en el sistema operatiu Linux.
- L'aplicació serà desenvolupada amb el llenguatge de programació C++.

- L'aplicació serà desenvolupada utilitzant la llibreria de gràfics *OpenGL*.

### 3.1.2 ESPECIFICACIÓ

En la fase d'especificació s'assignen les tasques que pot realitzar cada usuari del sistema a partir de les dades recollides en la fase d'anàlisi de requisits. El mètode clàssic de mostrar aquesta informació és mitjançant un diagrama de casos d'ús. En la figura 3-1 es pot veure el diagrama de casos d'ús de la aplicació.

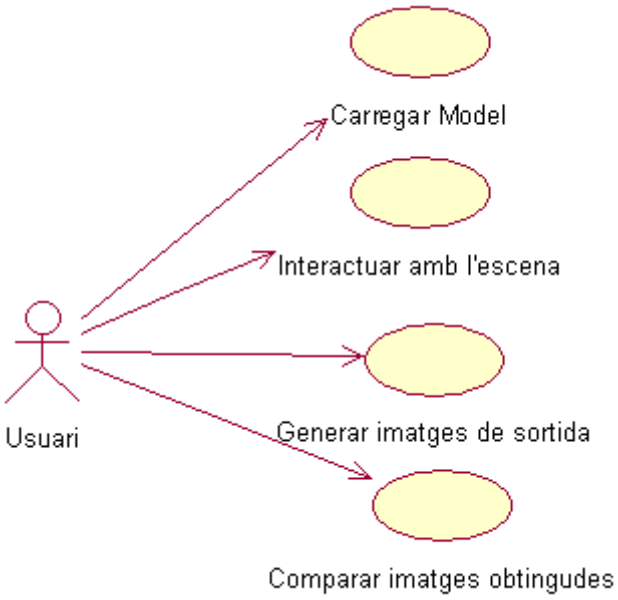



**Figura 3-1**

- Carregar el model: Permet a l'usuari escollir un model de tots els existents per tal de mostrar-lo a través de la aplicació.
- Interactuar amb l'escena: L'usuari, mitjançant els perifèrics d'entrada (teclat i ratolí) pot navegar per l'escena, canviant el punt de vista de la càmera, realitzant zoom, etc.
- Generar les imatges de sortida: Mitjançant la aplicació, l'usuari pot generar les imatges resultants d'aplicar les diferents tècniques sobre l'objecte carregat en el programa.

- Comparar les imatges obtingudes: Un cop generades les imatges, l'usuari pot comparar les imatges i l'aplicació mostrarà per pantalla els resultats.

### 3.1.3 ANÀLISI

En la fase d'anàlisi es dissenya un esquema de les entitats de software que seran implementades per tal de que la aplicació pugi dur a terme els casos d'ús descrits en la fase d'especificació. Al ser una aplicació petita, el diagrama de classes, que és la representació gràfica de l'esquema, és força senzill (figura 3-2) és molt simple i a continuació s'expliquen les classes que tenen una importància destacada.

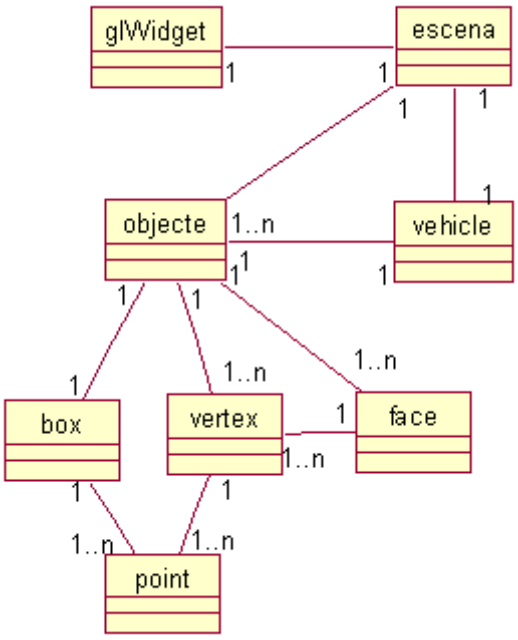



**Figura 3-2**

- *GlWidget*: Aquesta classe és la que s'encarrega de la major part de les tasques. Primer de tot, és una classe *Boundary* cosa que vol dir que és la que s'ocupa de gestionar la comunicació amb l'usuari mitjançant botons, comades de teclat, etc. També es la classe encarregada d'actualitzar la finestra que mostra l'objecte per mostrar els canvis que es puguin realitzar.

- Objecte: En aquesta classe es gestionen totes les funcionalitats que tenen a veure amb l'objecte, tals com el pintat de l'objecte, el càlcul de colors, etc.

La resta de classes tenen tasques menors que no és necessari comentar ja que no tenen la mateixa importància que les classes explicades anteriorment.

### 3.1.4 TECNOLOGIES UTILITZADES

A continuació es descriuen totes les tecnologies i llibreries de tercers utilitzades en la implementació de l'aplicació. Per a cadascuna d'elles, s'explica breument quina ha estat la seva finalitat.

- *OpenGL*: És una llibreria per a gestionar gràfics tant en 2D com en 3D. Implementa totes les funcions habituals que s'utilitzen quan s'utilitzen gràfics: transformacions geomètriques, projecció de l'escena a partir d'una càmera, càlcul del color dels objectes a partir de la il·luminació, etc.
- GLSL*:* És el llenguatge *d'OpenGL* per la programació en targetes gràfiques. Permet la modificació del *pipeline* de pintat *d'OpenGL.* Per poder fer ús de GLSL cal una targeta gràfica programable.
- GLew i GLut: Són llibreries que faciliten treballar amb extensions *d'OpenGL.* Una extensió és una funcionalitat afegida a les ja existents.
- Qt4: Donat que *OpenGL* no ofereix un sistema senzill per a generar interfícies d'usuari, s'ha buscat una alternativa. Qt4 permet, mitjançant l'aplicació *QtDesigner*, dissenyar d'una manera visual les interfícies d'usuari.

### 3.1.5 DETALLS D'IMPLEMENTACIÓ

En aquest apartat es descriuen decisions de disseny a l'hora de la implementació de l'algorisme.

Primer de tot, els objectes 3D de les escenes han de complir un requisit, que cada element de l'objecte estigui separat en *object groups.* Els *object groups* són agrupacions de parts de l'objecte global i són utilitzats per poder, entre d'altres, pintar tan sols *l'object group* que interessi en el moment.

Quan es carrega el model, per a cada grup es genera una *display list*. Una *display list* és un conjunt d'instruccions *OpenGL* que permet reduir el temps de pintat dels objectes. A més, a cada grup se li assigna un color generat aleatòriament que és el que es mostra en pantalla.

Una alternativa a les *display list* és utilitzar *vertex array*, que consisteix en emmagatzemar tota la informació del model en la memòria de la targeta gràfica, obtenint un temps de pintat fins i tot millor que el de les *display list*. L'aplicació d'aquesta tècnica va generar diversos problemes en els colors dels objectes i finalment es va decidir descartar aquesta opció.

Finalment s'ha optat per no incloure il·luminació i que el color de tots els objectes sigui homogeni, tot i perdre la percepció de volum que ens proporciona una bona il·luminació.

## 3.2 DEPTH PEELING

### 3.2.1 DESCRIPCIÓ

Tal i com s'ha explicat, l'ordre en el qual es pinten els objectes és important per a la representació dels objectes transparents. El mètode de *depth peeling*, explicat per C. Everitt, 2001, s'encarrega de mostrar en varies imatges diferents capes de profunditat del model de forma decreixent (de la més propera al punt de vista a la més llunyana). S'ha triat aquest mètode perquè, a més a més de l'ordenació, proporciona informació sobre el nombre de capes de cada píxel (és a dir, el nombre d'objectes que s'hi projecten).

En el mètode de pintat d'*OpenGL*, per a cada píxel de la pantalla hi ha projectat els objectes amb profunditat 0, és a dir, es projecta al píxel (x,y) l'objecte més proper al punt de vista. Això però, no és suficient.

Per a l'obtenció de les diferents capes, cal utilitzar *depth maps*. Tal i com funciona el sistema de pintat *d'OpenGL* (quan s'activa el depth test), al renderitzar cadascun dels objectes, es compara la profunditat de l'objecte actual amb la que ja hi ha al *framebuffer*. Si aquesta profunditat és menor, s'actualitza el color del *framebuffer* i el *buffer* que guarda la profunditat de cadascun dels píxels. Amb aquest sistema només obtenim el color de l'objecte de menor profunditat.

Els *depth maps* són *buffers* en els quals en cada element es guarda un component de profunditat. En el cas que ens ocupa, el *buffer* té la mateixa mida que el *viewport* i per tant, en cada element s'emmagatzema el component de profunditat del píxel corresponent.

### 3.2.2 IMPLEMENTACIÓ

Per a la correcta implementació del *Depth peeling*, primer cal destacar les eines que s'han utilitzat per aquest procés:

- *Shaders*: Els *shaders* han sigut implementats en GLSL i són els encarregats de treballar amb els *depth maps*.
- Per a accedir a les imatges, modificar-les i salvar-les s'ha utilitzat la classe QImage de Qt.
- Per a poder obtenir els resultats, cal configurar 2 *framebuffer objects*. El primer d'ells s'encarrega d'emmagatzemar el color de l'escena i mostrarà els resultats a l'usuari. El segon d'ells tan sols emmagatzema el component de profunditat de l'escena i la seva finalitat és la de crear el *depth map*.

Primer de tot, s'explica quina funció realitzen els *shaders*.

- *Vertex shader*: el *vertex shader* es calcula per a cada vèrtex que s'envia a pintar, i bàsicament transforma el punt de coordenades d'objecte a coordenades de pantalla.
- *Fragment shader*: aquest *shader* s'executa per a cada fragment que es pintarà a pantalla. S'encarrega de la part més important d'aquesta aplicació, ja que per a cada fragment accedeix a la textura que conté el *depth map* a les coordenades que corresponen al fragment i compara ambdues profunditats. En el cas de que la seva profunditat (coordenada z) sigui menor que la que conté el *depth map* es descartarà el fragment, altrament, es pintarà el fragment amb el color corresponent.

Un cop definides les funcions que realitzen els *shaders*, es pot tractar en profunditat el funcionament de la resta de l'algorisme de *Depth peeling*. Per a la correcta realització del mètode cal pintar dos cops l'escena sencera per a cada capa. Els passos a realitzar són els següents per obtenir cada capa:

1. Configurar l'aplicació per tal de que pinti en el *framebuffer* que emmagatzema el color i carregar en la textura que utilitza el fragment *shader* el *depth map* obtingut en la iteració anterior. En el cas que sigui la primera iteració, aquesta textura no es té en compte.

2. Pintar l'escena completa i emmagatzemar el resultat en la posició corresponent d'un vector de QImage, on es troben guardades les diferents capes obtingudes.
3. Configurar l'aplicació per tal de que pinti en el *framebuffer* que emmagatzema el component de profunditat de l'escena i carregar el *depth map* de la iteració anterior a la textura que utilitza el fragment *shader*. En el cas que sigui la primera iteració, aquesta textura no es té en compte.
4. Pintar l'escena completa i emmagatzemar el resultat a la textura que conté el *depth map*, substituint a la que hi havia anteriorment.

Al finalitzar l'algorisme, es té en un vector les imatges corresponents a cada capa i es pot accedir individualment a cadascuna per a calcular més tard els diferents mètodes de composició de color.

### 3.2.3 EXEMPLES

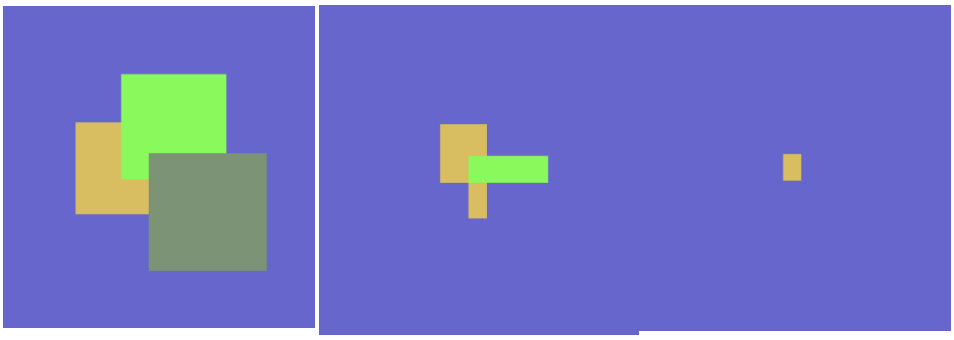


**Figura 3-3. *Depth peeling* de profunditat 3 on es poden veure 3 rectangles.**

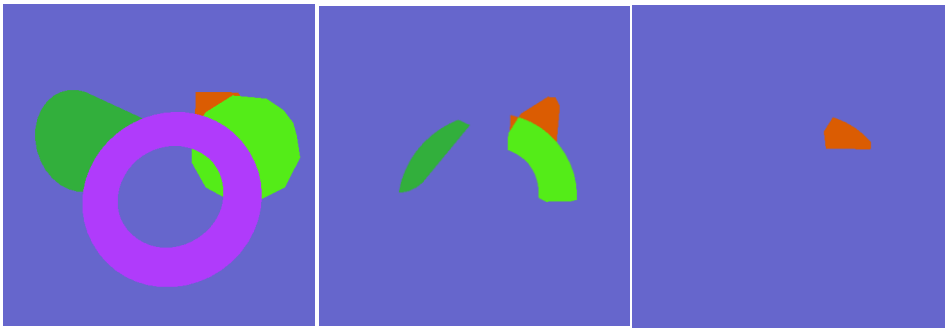


**Figura 3-4. *Depth peeling* de profunditat 3 on es pot veure un torus, un con, una esfera i un cub.**

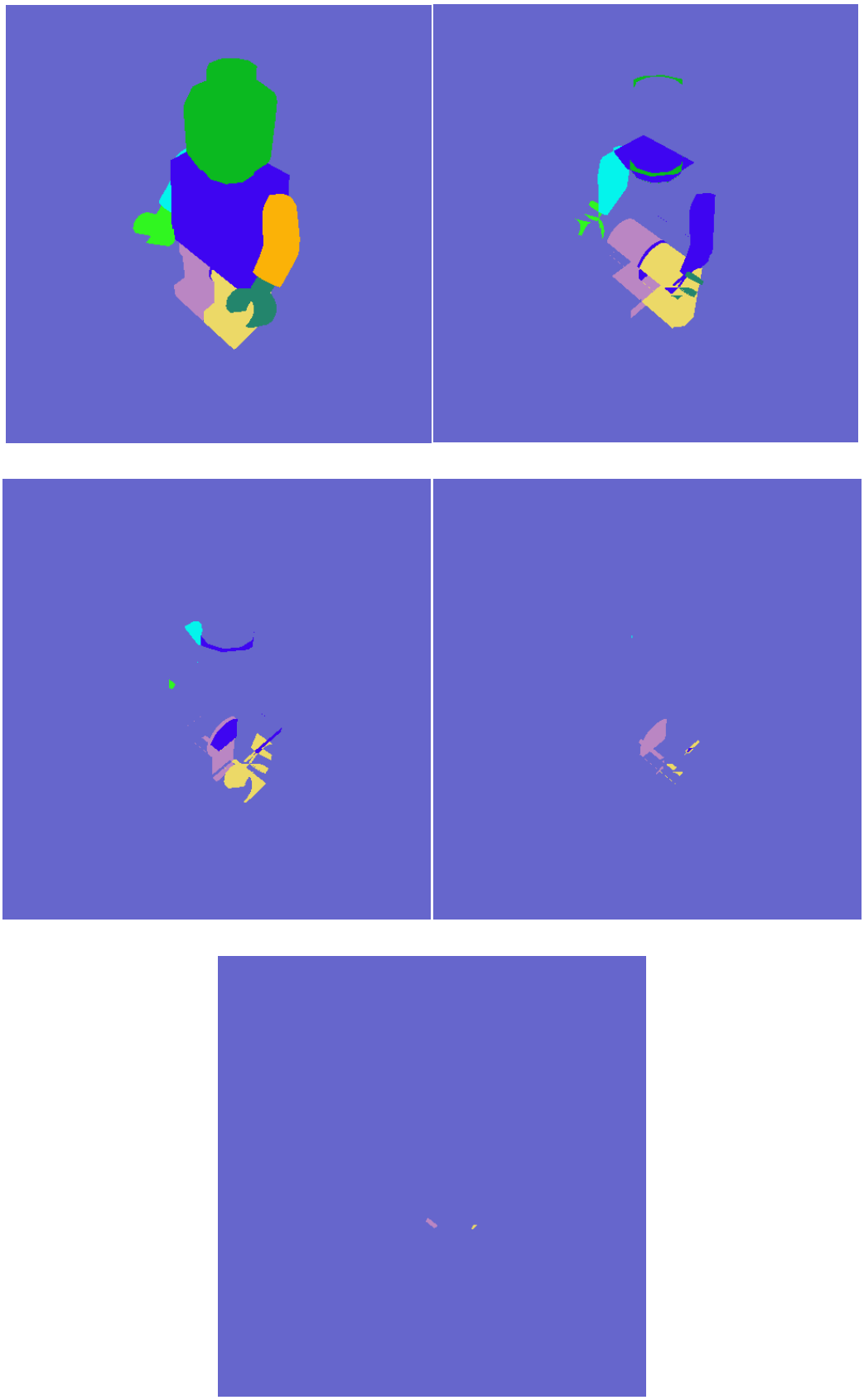


**Figura 3-5. *Depth peeling* de profunditat 5 d'una figura de Lego.**

## 3.3 TÈCNIQUES

### 3.3.1 DESCRIPCIÓ

S'han desenvolupat un conjunt de 6 tècniques que enfoquen el problema per diferents bandes. Primer de tot, s'ha implementat la tècnica d'opacitat uniforme, que dóna a tots els objectes la mateixa opacitat. S'ha escollit aquesta tècnica perquè a part de ser la solució que s'escull en aquests casos, serveix per a comparar si hi ha millores en les solucions proposades sobre aquesta.

La resta de tècniques es poden desglossar en 3 grups:

- Opacitat per pixel depenent de la capa: Les tècniques que entren en aquest grup utilitzen les imatges obtingudes a partir del *depth peeling*. A cada una de les capes, s'aplica una opacitat diferent, donant en uns casos més importància a les capes més properes i en altres, a les més llunyanes. Els noms de les tècniques d'aquest grup són:
  - Opacitat decreixent per capa (per píxel).
  - Opacitat creixent per capa (per píxel)
- Opacitat per objecte tenint en compte les capes: Les 2 tècniques que exploren aquesta metodologia assignen a cada objecte una capa llavors, segons la capa on està assignat, se li aplica una opacitat diferent. L'assignació de capa no és trivial, perquè es necessita saber en quines capes es troba cada objecte.
  - Opacitat decreixent per capa (per objecte).
  - Opacitat creixent per capa (per objecte).
- Optimització d'opacitat per objecte: La última tècnica consisteix en un algorisme que assigna una opacitat diferent a cada objecte intentant maximitzar el nombre d'objectes ben contrastats a la imatge final.

#### 3.3.1.1 COLOR CODING

Amb les eines descrites fins al moment, no és possible d'implementar totes les tècniques descrites ja que les tècniques d'opacitat per objecte necessiten conèixer per a cada objecte, en quines capes es troba representat.

Trobar aquesta informació no és trivial, ja que *OpenGL* no ofereix cap mètode que permeti obtenir les dades necessàries, per tant, ha estat necessari idear un algorisme que generi tota la informació. Un cop més es fa ús de l'algorisme de *depth peeling* per a generar les diferents capes, però s'han fet petites variacions per tal d'obtenir el resultat desitjat.

Per a cada objecte de l'escena, es codifica el seu identificador d'objecte en un color únic i tot l'objecte té aquest color calculat. La codificació s'obté a partir d'una permutació dels 24 bits disponibles pel color: 8 pel color vermell, 8 pel color verd i 8 pel color blau. La codificació segueix la següent fórmula, que calcula la representació de l'identificador en base 256:

- $Component\ vermell = identificador/256^2$
- $Component\ verd = (identificador\%256^2)/256$
- $Component\ blau = (identificador\%256^2)\%256$

Quan es vol descodificar el color, la fórmula a seguir per obtenir l'identificador de l'objecte és la següent:

$$Identificador = component\ blau + 256 \cdot component\ verd + 256^2 \cdot component\ vermell$$

Un cop obtingut el resultat d'aplicar el *depth peeling* a l'escena amb aquests colors, només cal recórrer tots els píxels de totes les capes descodificant els colors obtinguts per a saber quin objecte es troba projectat en cada píxel i emmagatzemar els resultats en una llista de vectors on cada llista és un objecte, mentre que el contingut del seu vector és el conjunt de capes on s'ha trobat aquest objecte.

## 3.3.2 OPACITAT UNIFORME

### 3.3.2.1 DESCRIPCIÓ

La primera tècnica que s'ha implementat ha estat la *d'alpha blending* amb d'opacitat uniforme. Aquesta tècnica, com s'ha comentat en apartats anteriors, és la més utilitzada actualment i serveix com a base per a comparar amb les diferents solucions proposades en aquest projecte.

Tal i com s'ha explicat en l'apartat 2, el mètode de *alpha blending* consisteix en aplicar a cada objecte una opacitat i composar el color final seguint la següent fórmula (en el cas de 2 objectes solapats):

$$Color = Color_1 \cdot \alpha + (1 - \alpha) \cdot Color_2$$

Del que es pot deduir de la fórmula, es veu que els objectes més propers a l'observador tenen una contribució més important al color, però en canvi, en quant als objectes que es troben a una profunditat de vàries capes la seva contribució és mínima, per tant, aquests objectes són més complicats, sinó impossibles, d'identificar.

### 3.3.2.2 IMPLEMENTACIÓ

Per a implementar aquesta tècnica es poden escollir dues opcions. La més senzilla és utilitzar les funcions que ofereix *OpenGL* per a produir aquest resultat. Tot i això, s'ha decidit descartar aquesta opció i implementar aquesta tècnica fent servir *depth peeling*.

Per altra banda, s'ha decidit d'utilitzar una *alpha* uniforme per a tots els objectes de 0.5, obtenint així una opacitat igual per a cada objecte. S'ha pres aquesta decisió degut a que valors més grans només mostren les primeres capes de la vista, mentre que valors més petits fan que predomini el color de fons.

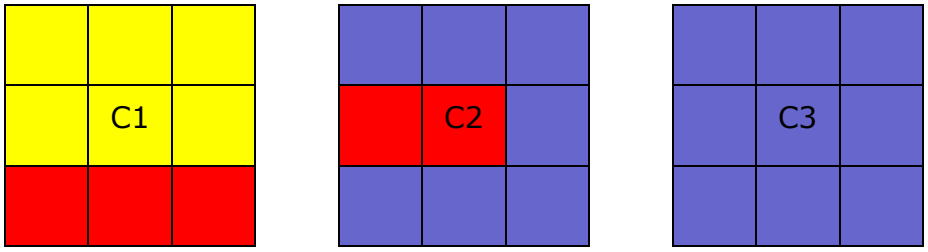
En la implementació d'aquesta solució, com en d'altres que es veuran en altres apartats es fa ús del conjunt d'imatges obtingudes mitjançant la tècnica de *depth peeling*. Els passos a realitzar són els següents:

1. Generar una imatge de la mida correcta.
2. Per a cada píxel de la imatge:
    1. Accedir al píxel corresponent a la imatge de la capa més llunyana menys 1 a l'usuari, ignorant la capa més llunyana.
    2. Aplicar la fórmula com si només hi haguessin 2 capes, és a dir, el color es calcula de la manera que es veu a continuació:

$$Color = Color\ 1\ \cdot\ \alpha + (1 - \alpha) \cdot Color\ 2$$

    El color 1 és el color del píxel que es troba en la capa que s'està mirant. En el cas que sigui la primera iteració, el color 2 és el que conté el píxel corresponent a la capa més llunyana. Altrament, el color 2 és el que ha estat calculat en la iteració anterior.
    3. Emmagatzemar el color resultant per tal de que s'utilitzi en la següent iteració.
    4. Repetir els passos 1-3 utilitzant cada cop una capa més propera a l'usuari.

### 3.3.2.3 EXEMPLES



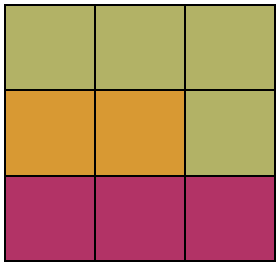

En aquest exemple, es poden veure 3 imatges corresponents a varies capes, de més propera a més llunyana. Per a veure el funcionament de l'algorisme, només es tindrà en compte els píxels marcats com a C1, que el seu codi RGB és (255,255,0); C2, amb codi (255,0,0) i C3, amb el codi corresponent a (102,102,204). El resultat al aplicar l'algorisme quedaria de la següent forma:

- Iteració 1, la primera capa que es visita és la segona, per tant, en la fórmula *d'alpha blending*, el color 1 és C2 mentre que el color 2 és C3. Al aplicar la fórmula el resultat és de:
  - Component Vermell = 0.5 * 255 + (1 - 0.5) * 102 = 178
  - Component Verd = 0.5 * 0 + (1 - 0.5) * 102 = 51
  - Component Blau = 0.5 * 0 + (1 – 0.5) * 204 = 102

  Així doncs, el color resultant que s'emmagatzema per utilitzar com a color 2 per a la següent iteració es el que té com a codi RGB igual a (178,51,102).

- Iteració 2, s'accedeix a la primera capa i com a color 1 es selecciona C1, i com a color 2 el color obtingut anteriorment. Com que aquesta és la última capa, el color obtingut és el color final que tindrà el píxel després d'aplicar l'algorisme. A continuació és mostra el resultat:
  - Component Vermell = 0.5 * 255 + (1 – 0.5) * 178 = 216
  - Component Verd = 0.5 * 255 + (1 – 0.5) * 51 = 153
  - Component Blau = 0.5 * 0 + (1 – 0.5) * 102 = 51

En la propera imatge és pot veure el resultat de l'algorisme després de calcular el resultat per cada píxel:



En la figura 3-6 es pot veure el resultat d'aplicar la tècnica d'opacitat uniforme a uns models simples amb una profunditat de 15 capes.

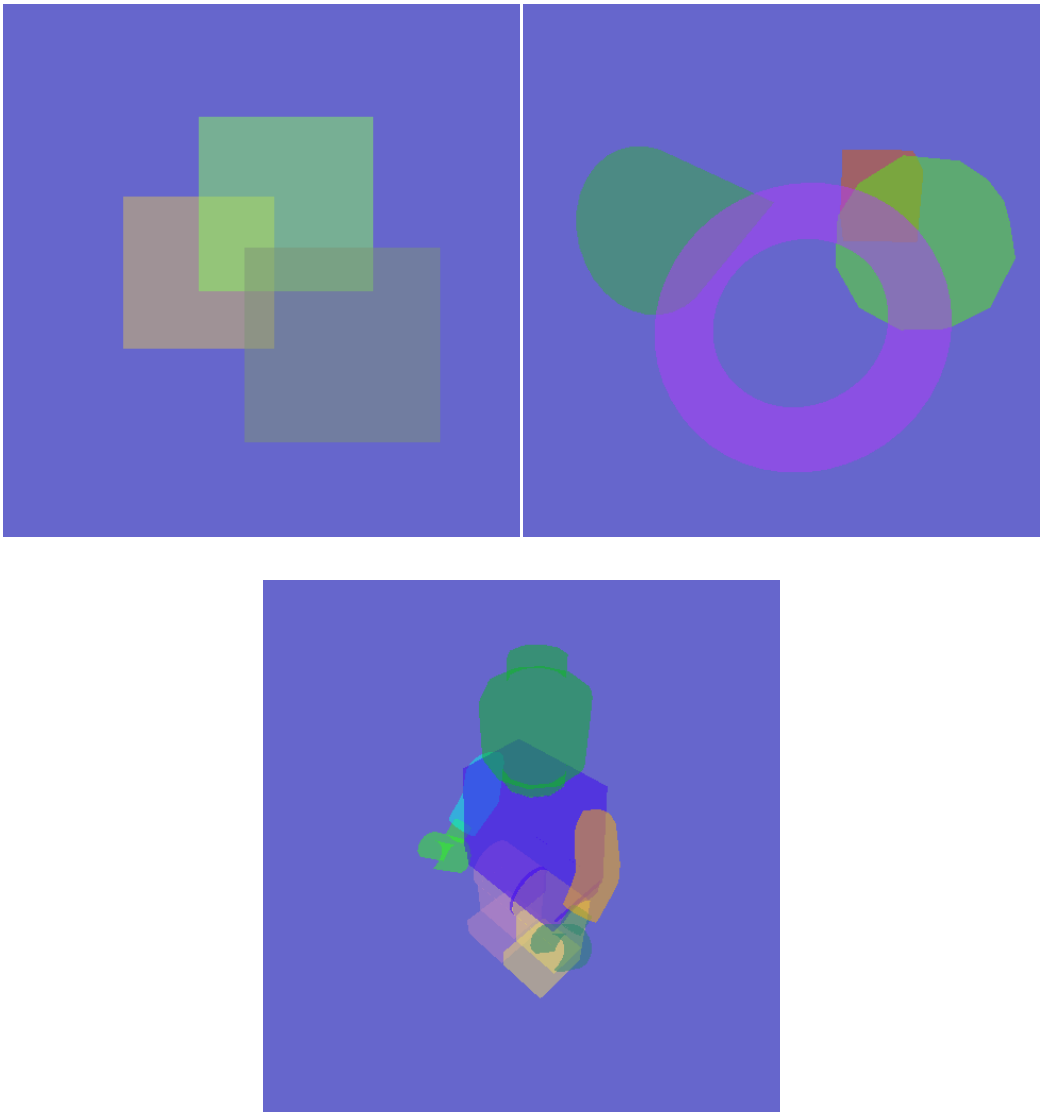


**Figura 3-6**

## 3.3.3 OPACITAT DECREIXENT PER CAPA (PER PÍXEL)

### 3.3.3.1 DESCRIPCIÓ

Una de les tècniques més senzilles que es pot dissenyar és la que consisteix en assignar a cada píxel de cada capa una opacitat diferent. En aquest cas, els valors d'opacitat es tractaran com si fossin pesos, de manera que el color resultant en un píxel es calcularà com la mitja ponderada dels colors dels objectes que s'hi projecten. Per tal de simular un comportament de la llum més realista, on els objectes llunyans tenen una contribució menor que els objectes propers, una primera opció consisteix a assignar una opacitat decreixent segons la capa, és a dir, s'assigna una opacitat major a les capes properes a la càmera i una opacitat menor a les més llunyanes.

Per a l'assignació d'opacitat es té en compte el nombre de capes que s'han creat mitjançant l'aplicació de l'algorisme de *depth peeling* i es dona a cada píxel de cada capa una opacitat igual a (n – i ) / n, sent **i** l'índex de la capa i **n** el nombre de capes total.

Cal tenir en compte els casos en que el en el cas de que el píxel (x,y) en una gran quantitat de capes és color de fons la contribució d'aquest color seria massa gran mentre que el color dels objectes seria molt reduïda. Per tant, s'ha decidit que només es tingui en compte el color de fons en el cas de que en totes les capes el píxel (x,y) sigui color de fons.

### 3.3.3.2 IMPLEMENTACIÓ

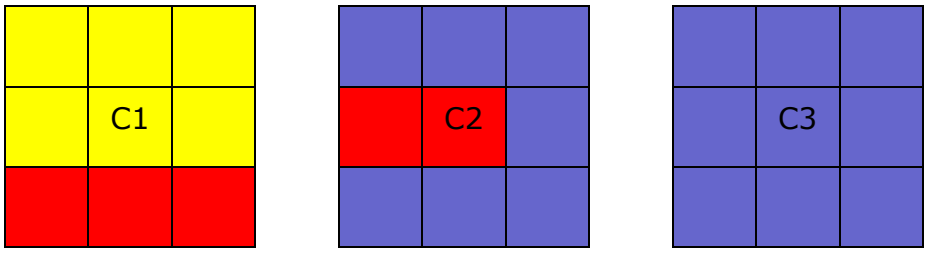
Per a la implementació d'aquesta tècnica només cal tenir en compte dos aspectes, la opacitat que li correspon a cada capa i tenir cura amb la contribució al color final del color de fons, per tant, per a calcular el color de cada píxel de la imatge es segueixen els següents passos:

1. Inicialitzar un nou color S amb els components R,G i B igual a 0.
2. Accedir a la capa més propera a l'usuari i calcular la opacitat de la capa seguint la següent fórmula, sent **i** l'índex de la capa i **n** el nombre total de capes:

$$Opacitat = (n - i)/n$$

3. Comprovar que el píxel no sigui color de fons, si no ho és, multiplicar el color del píxel per la opacitat corresponent a la seva capa i es suma al color S. Per tant S anirà mantenint la suma ponderada dels objectes que es projecten a cada pixel.
4. Repetir els passos 2 i 3 cada cop amb una capa més llunyana.
5. En el cas que el color inicialitzat en el pas 1 no hagi estat modificat, vol dir que el píxel seleccionat en totes les capes és color de fons, per tant, en el color S s'emmagatzema el color de fons. Altrament, normalitzar el color resultant dividint cada component del color S per la suma d'opacitats utilitzades per obtenir un color en el rang 0-255.
6. Emmagatzemar el color calculat en el píxel corresponent de la imatge de sortida.

#### 3.3.3.3 EXEMPLES




En aquest exemple es pot veure el resultat d'aplicar l'algorisme de *depth peeling* de profunditat 3. Per a veure el resultat d'aplicar la tècnica actual només es tindran en compte els píxels identificats com a C1, amb codi RGB igual a (255,255,0), C2 amb codi (255,255,0) i C3 amb el codi corresponent a (102,102,204). El color de fons d'aquesta escena és el que correspon al codi RGB (102,102,204). A continuació es mostra el resultat d'aplicar l'algorisme explicat anteriorment:

- S'inicialitza el color de sortida S amb codi RGB igual a (0,0,0).
- Iteració 1.

- Es calcula la opacitat corresponent a la capa actual. La capa actual té l'índex 0 i s'han obtingut 3 capes, per tant, la opacitat és igual a (3 – 0) / 3 = 1.
- Es comprova que C1 sigui diferent del color de fons, com ho és, es calcula el seu valor que s'afegeix al color de sortida S:
  - Component vermell = 255 * 1 = 255
  - Component verd = 255 * 1 = 255
  - Component blau = 0 * 1 = 0

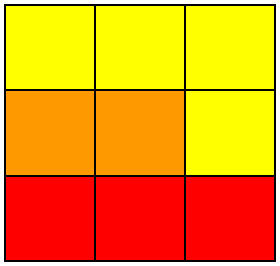
  Al afegir el resultat al color de sortida, S té actualment el codi RGB igual a (255,255,0).

- Iteració 2.
  - Es calcula la opacitat corresponent a la capa actual. L'índex de la capa és 1 i s'han obtingut 3 capes, per tant, la opacitat és de (3 – 1) / 3 = 2/3.
  - Es comprova que C2 sigui diferent del color de fons, al ser-ho, es calcula el valor que s'afegeix al color de sortida S:
    - Component vermell = 255 * 2/3 = 170
    - Component verd = 0 * 2/3 = 0
    - Component blau = 0 * 2/3 = 0

    Afegint aquest resultat a S, el seu nou valor és de (425,255,0).
- Iteració 3.
  - Es calcula la opacitat corresponent a la capa 2. Aquesta opacitat és de (3 -2)/3 = 1/3.
  - Com que C3 és color de fons, finalitza la iteració.
- Finalment, cal normalitzar els colors de S. Per tant, cal dividir cada component per la suma d'opacitats que s'han utilitzat (1 + 2/3):
  - Component vermell = 425/(5/3) = 255
  - Component verd = 255/(5/3) = 153
  - Component blau = 0/(5/3) = 0

  Per tant, el color definitiu S és el que correspon al codi RGB (255,153,0).

En la següent imatge es pot veure el resultat d'aplicar l'algorisme per a cada píxel:



En la figura 3-7 es poden veure uns quants exemples del resultat de l'aplicació de l'algorisme en escenes amb una profunditat de 15 capes.

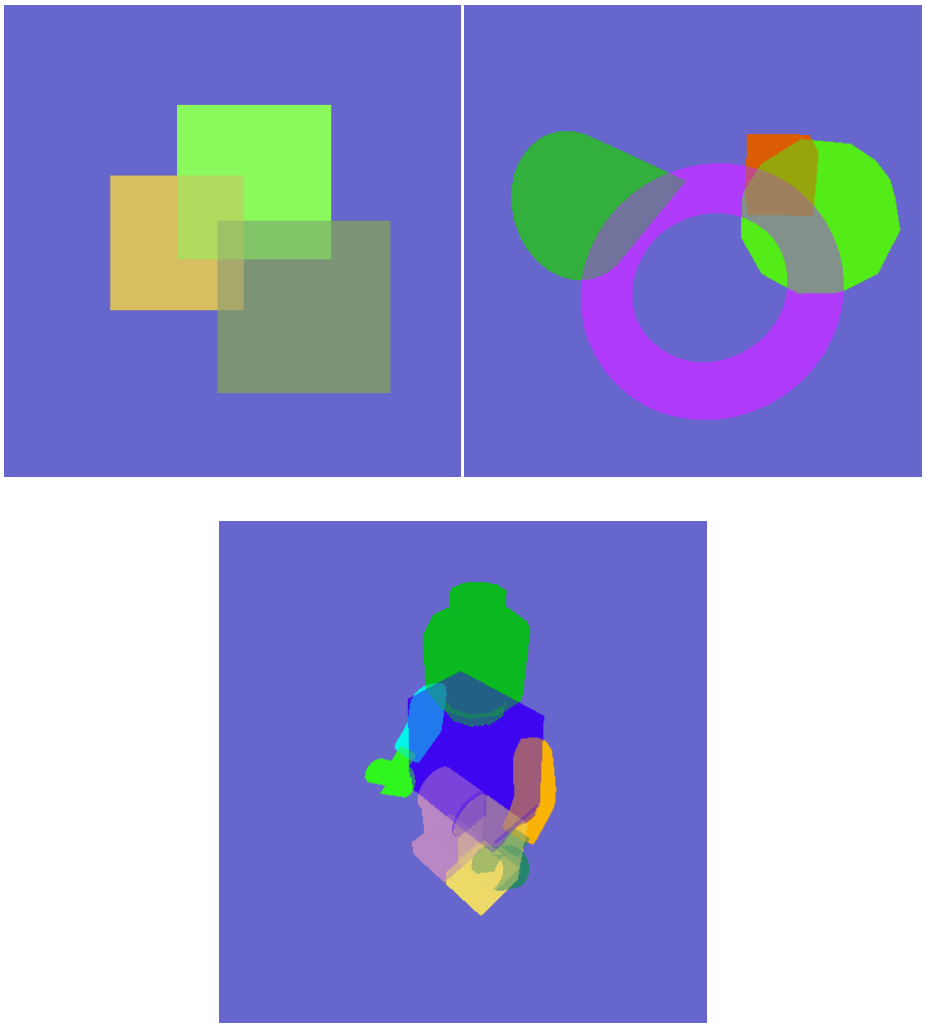


**Figura 3-7**

### 3.3.4 OPACITAT CREIXENT PER CAPA (PER PÍXEL)

#### 3.3.4.1 DESCRIPCIÓ

Per a les escenes amb molts nivells d'oclusió, la tècnica de opacitat per capa decreixent en profunditat té el mateix tipus de problema que la tècnica d'opacitat uniforme: els objectes que es troben a varis nivells de profunditat contribueixen de manera molt petita al color final, fent que siguin molt difícils d'identificar.

La tècnica descrita en aquest apartat intenta corregir aquest problema assignant una opacitat per capa, però en aquest cas serà menor si es troba a prop de la càmera i major si es troba més lluny. La opacitat que s'assigna a cada capa segueix la següent fórmula: (i + 1) / n, essent **i** el número de capa actual i **n** el nombre de capes total.

Al donar més importància les capes més llunyanes, la imatge resultant només millora l'aspecte de un nombre d'elements identificables, mentre que dificulta la identificació espaial dels mateixos.

Finalment, en aquest algorisme tampoc es té en compte la contribució del color de fons, perquè tal i com passava en la tècnica anterior, en escenes amb una profunditat gran on hi ha varies capes amb només color de fons, la seva contribució seria tan gran que faria que els colors dels objectes amb prou feines es reconeguessin.

#### 3.3.4.2 IMPLEMENTACIÓ

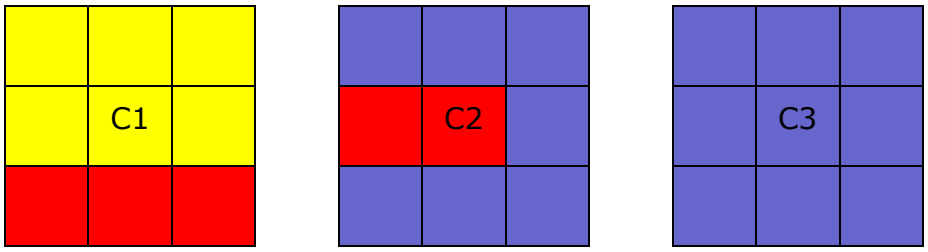
A continuació es descriuen els passos que s'han de seguir per a l'aplicació de l'algorisme per a cada píxel de l'escena:

1. Inicialitzar el color de sortida S amb el codi RGB igual a (0,0,0).
2. Accedir a la capa més propera a l'usuari i calcular la opacitat corresponent a la capa amb la següent fórmula, essent **i** l'índex de la capa i **n** el nombre de capes obtingudes a partir de l'algorisme de *depth peeling*:

$$Opacitat = (i + 1)/n$$

3. Comprovar que el color del píxel actual sigui diferent del color de fons. Si no ho es, s'acaba la iteració. Altrament, es multiplica el color del píxel per la opacitat de la capa i es suma el resultat al color de sortida S.
4. Repetir els passos 2 i 3 cada cop amb una capa més llunyana.
5. En el cas de que el color S no hagi estat modificat, posar el color S igual al color de fons, ja que vol dir que en el píxel donat no s'ha trobat en cap capa un color diferent a color de fons. En cas d'haver estat modificat, normalitzar el color S dividint cada component R, G i B per la suma de totes les opacitats utilitzades per obtenir un valor en el rang 0-255.
6. Emmagatzemar el color S en el píxel corresponent de la imatge de sortida.

#### 3.3.4.3 EXEMPLES




En aquest exemple es pot veure el resultat d'aplicar l'algorisme de *depth peeling* de profunditat 3 en una escena. Per a veure el comportament de la tècnica, ens centrarem en els píxels C1, amb codi RGB igual a (255,255,0), C2 que té codi (255,0,0) i C3 amb el codi (102,102,204). Com a color de fons es considera el color amb codi RGB (102,102,204). A continuació es mostra el funcionament de l'algorisme:

- S'inicialitza el color de sortida S amb el codi RGB (0,0,0).
- Iteració 1.

- Es calcula la opacitat corresponent a la capa amb índex 0. Tenint en compte que el nombre de capes totals és de 3 s'obté que la opacitat de la capa és igual a (0 + 1) / 3 = 1/3.
- Es comprova que el color del píxel sigui diferent del color de fons. Com ho és, es multiplica cada component de C1 per la opacitat de la capa:
  - Component vermell = 255 * 1/3 = 85
  - Component verd = 255 * 1/3 = 85
  - Component blau = 0 * 1/3 = 0

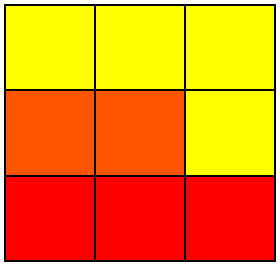
  Es suma el resultat de la multiplicació al color S quedant actualment amb el color (85,85,0).

- Iteració 2.
  - Es calcula la opacitat corresponent a la capa 1. Al haver 3 capes, la opacitat de la capa actual és de (1 +1 ) / 3 = 2/3.
  - Es comprova que el color del píxel sigui diferent del color de fons. Al ser-ho, es multiplica el color C1 per la opacitat de la capa:
    - Component vermell = 255 * 2/3 = 170
    - Component verd = 0 * 2/3 = 0
    - Component blau = 0 * 2/3 = 0

    Es suma el resultat al color S obtenint així el codi (255,85,0).
- Iteració 3.
  - S'obté la opacitat de la capa 2 que correspon a (2 + 1) / 3 = 1.
  - Es comprova que el color del píxel sigui diferent del color de fons. Com no ho és, s'acaba la iteració.
- Normalitzar el color de sortida dividint cada component de color per la suma de les opacitats utilitzades (1/3 + 2/3 = 1):
  - Component vermell = 255/1 = 255
  - Component verd = 85/1 = 85
  - Component blau = 0/1 = 0

  El color final que s'emmagatzema en la imatge final es el corresponent al codi RGB (255,85,0).

En la següent imatge es pot veure el resultat d'aplicar l'algorisme a tots els píxels de la imatge:



En la figura 3-8 es mostra un conjunt d'imatges d'escenes més complexes després d'aplicar l'algorisme amb una profunditat de 15 capes.

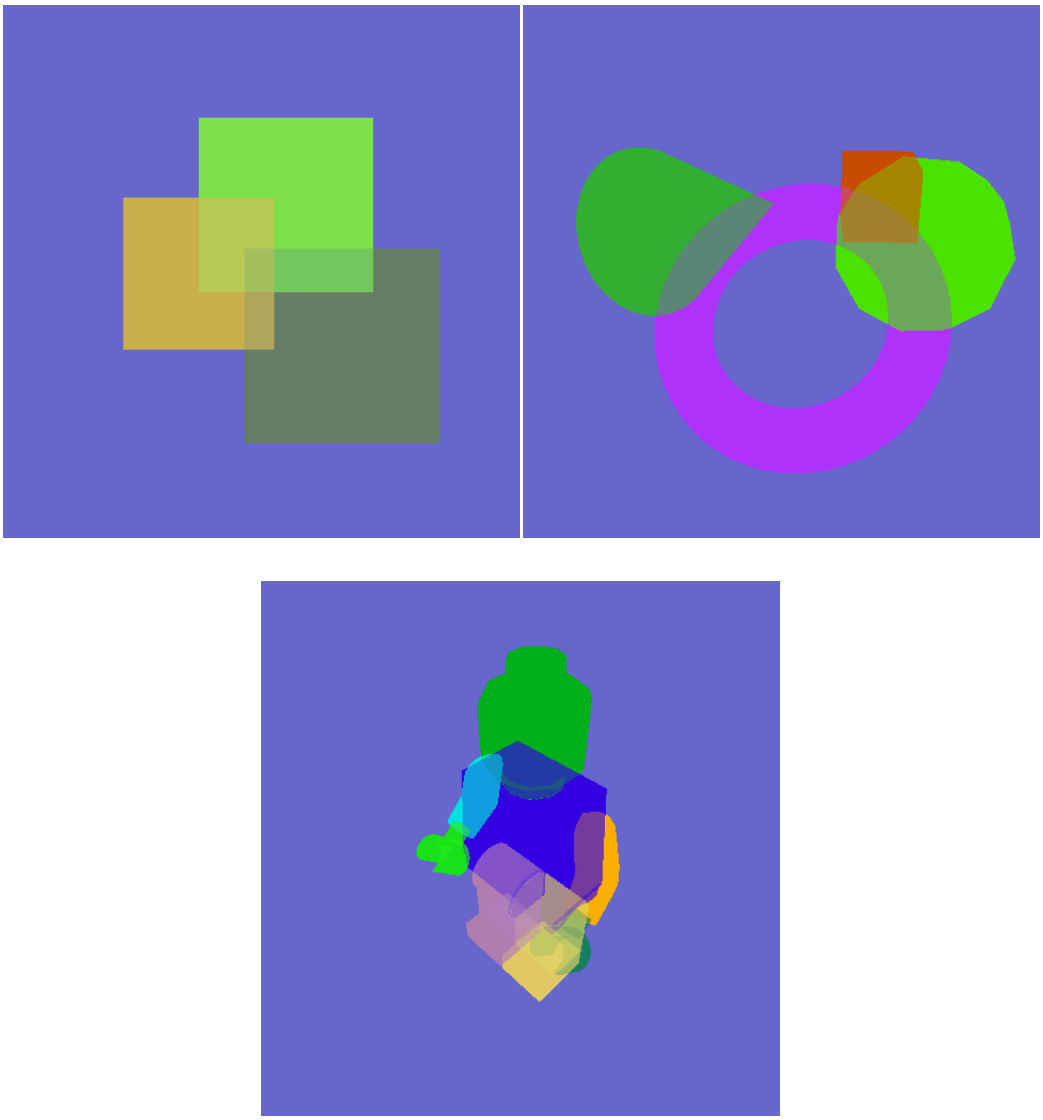


**Figura 3-8**

## 3.3.5 OPACITAT DECREIXENT PER CAPA (PER OBJECTE)

### 3.3.5.1 DESCRIPCIÓ

Una de les millores que es pot aplicar a les noves tècniques descrites fins al moment és l'aplicació d'una opacitat per objecte, enlloc de per píxel, ja que al donar una opacitat per píxel, es pot obtenir que un objecte tingui diferents opacitats en diferents regions de la pantalla, donant un resultat no del tot satisfactori. Per això, s'ha decidit assignar a cada objecte una opacitat seguint la següent fórmula: $(n-m)/n$, sent **n** el nombre de capes en les quals es troba l'objecte i **m** la mediana de les capes en les que es troba l'objecte. En resum, s'assigna l'objecte a una única capa i tot l'objecte té la opacitat corresponent a la capa calculada.

Per a poder utilitzar aquesta tècnica cal tenir informació extra que fins al moment no era necessària: conèixer en quines capes es troba cada objecte i donat un píxel, saber quins objectes es troben projectats en el mateix i en quin ordre estan. Gràcies a la tècnica de *color coding* explicada en l'apartat 4.2.1.1 podem trobar aquesta informació.

Lògicament, el color de fons no té el comportament d'un objecte i per tant només apareix a la imatge final si en el píxel corresponent no hi ha cap objecte projectat, és a dir, que mai contribueix en el color final dels objectes.

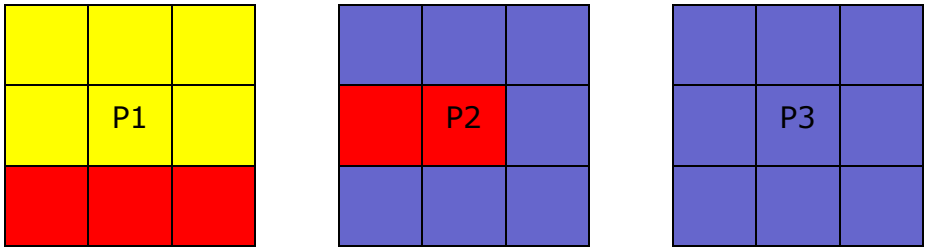
#### 3.3.5.2 IMPLEMENTACIÓ

El disseny pràctic d'aquesta tècnica és senzill però s'ha de tenir cert control dels valors que s'utilitzen en la fórmula per a obtenir la opacitat. En alguns casos, com per exemple que la mediana tingui el mateix valor que el nombre de capes en les quals apareix l'objecte, la fórmula no dona el comportament esperat i s'han hagut d'ajustar els resultats.

Un cop es tenen en compte aquests detalls, el comportament de l'algorisme segueix els següents passos:

1. A partir de la tècnica de *color coding* obtenir per a cada objecte en quines capes es troba representat.
2. Per a cada objecte calcular la mediana **m** de les capes on es troba. Aquesta serà la capa a la que queda assignat cada objecte.
3. Per a cada píxel de la imatge:
    a. Inicialitzar el color de sortida S amb codi RGB (0,0,0).
    b. Accedir a la capa més propera a l'usuari.
    c. Comprovar que el color sigui diferent del color de fons. Si ho és, obtenir l'objecte que correspon al color. Altrament, finalitzar la iteració.
    d. Calcular la opacitat de l'objecte mitjançant la fórmula:
    $$Opacitat = (n - m)/n$$
    e. Calcular el color a sumar a S multiplicant el color del píxel per la seva opacitat.
    f. Sumar el color obtingut a S.
    g. Repetir els passos **b** a **f** cada cop amb una capa més llunyana.
    h. Normalitzar el color S dividint cada component per la suma d'opacitats utilitzades per obtenir un codi en el rang 0-255. En el cas de que S no hagi estat modificat, emmagatzemar en S el color de fons.
    i. Emmagatzemar el color S en el píxel corresponent de la imatge de sortida.

### 3.3.5.3 EXEMPLES




En aquest exemple es pot veure el resultat d'aplicar *depth peeling* en una escena amb profunditat 3. Per a veure el funcionament de l'algorisme, s'utilitzaran els píxels corresponents a P1 de l'objecte 0, que té codi RGB (255,255,0), P2, que pertany a l'objecte 1 i té codi corresponent a (255,0,0) i P3, que és color de fons amb el codi (102,102,204). A continuació es mostren els passos que es segueixen:

- Per a cada objecte s'identifiquen les capes on es troben representats:
  - Objecte 0: Capa 0.
  - Objecte 1: Capa 0, Capa 1.
- Per a cada objecte es calcula la mediana de les capes on es troba:
  - Objecte 0: Mediana = 0
  - Objecte 1: Mediana = 0.5
- S'inicialitza el color de sortida S.
- Iteració 1.
  - Es comprova que el color que es troba en el píxel P1 sigui diferent del color de fons.
  - Es descobreix que el color que es troba en el píxel P1 correspon a l'objecte 0 utilitzant la tècnica de *color coding* i es calcula la opacitat que tindrà el color que es troba en el píxel P1. L'objecte 1 està representat en 1 capa i la seva mediana és de 0, per tant, la opacitat és igual a (1 – 0) / 1 = 1.
  - Es multiplica el color de P1 per la opacitat de l'objecte:
    - Component vermell = 255 * 1 = 255
    - Component verd = 255 * 1 = 255

        - Component blau = 0 * 1  = 0

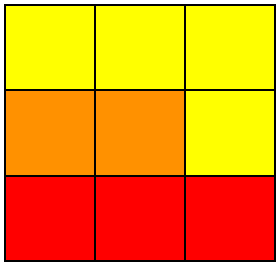
    Es suma el color calculat a S, el valor del qual passa a ser (255,255,0).

- Iteració 2.
    - Es comprova que el color que es troba en el píxel P2 sigui diferent del color de fons.
    - Es calcula que el color del píxel P2 correspon a l'objecte 1 mitjançant la tècnica de *color coding* i es calcula la opacitat corresponent. Donat que l'objecte 1 es troba en 2 capes i la seva mediana es 0.5 la seva opacitat és de (2 – 0.5) / 2 = 0.75.
    - Es multiplica el color de P2 per la opacitat trobada:
        - Component vermell = 255 * 0.75 = 191
        - Component verd = 0 * 0.75 = 0
        - Component blau = 0 * 0.75 = 0

    Es suma el color calculat a S, el valor del qual passa a ser (446,255,0).
- Iteració 3.
    - Es comprova que el color que es troba en el píxel P3 sigui diferent del color de fons. Com és igual, finalitza la iteració.
- Es normalitza el color S dividint cada component per la suma d'opacitats utilitzades (1 + 0.75 = 1.75)
    - Component vermell = 446 / 1.75 = 255
    - Component verd  = 255 / 1.75 = 145
    - Component blau = 0 / 1.75 = 0

  Per tant, el color final que s'emmagatzema a la imatge és el que correspon al codi RGB (255,145,0).

En la següent imatge es mostra el resultat de l'aplicació de l'algorisme en tots els píxels de la imatge:



En la figura 3-9 es poden veure varis exemples de l'aplicació de l'algorisme a varies escenes amb un nombre de capes igual a 15.

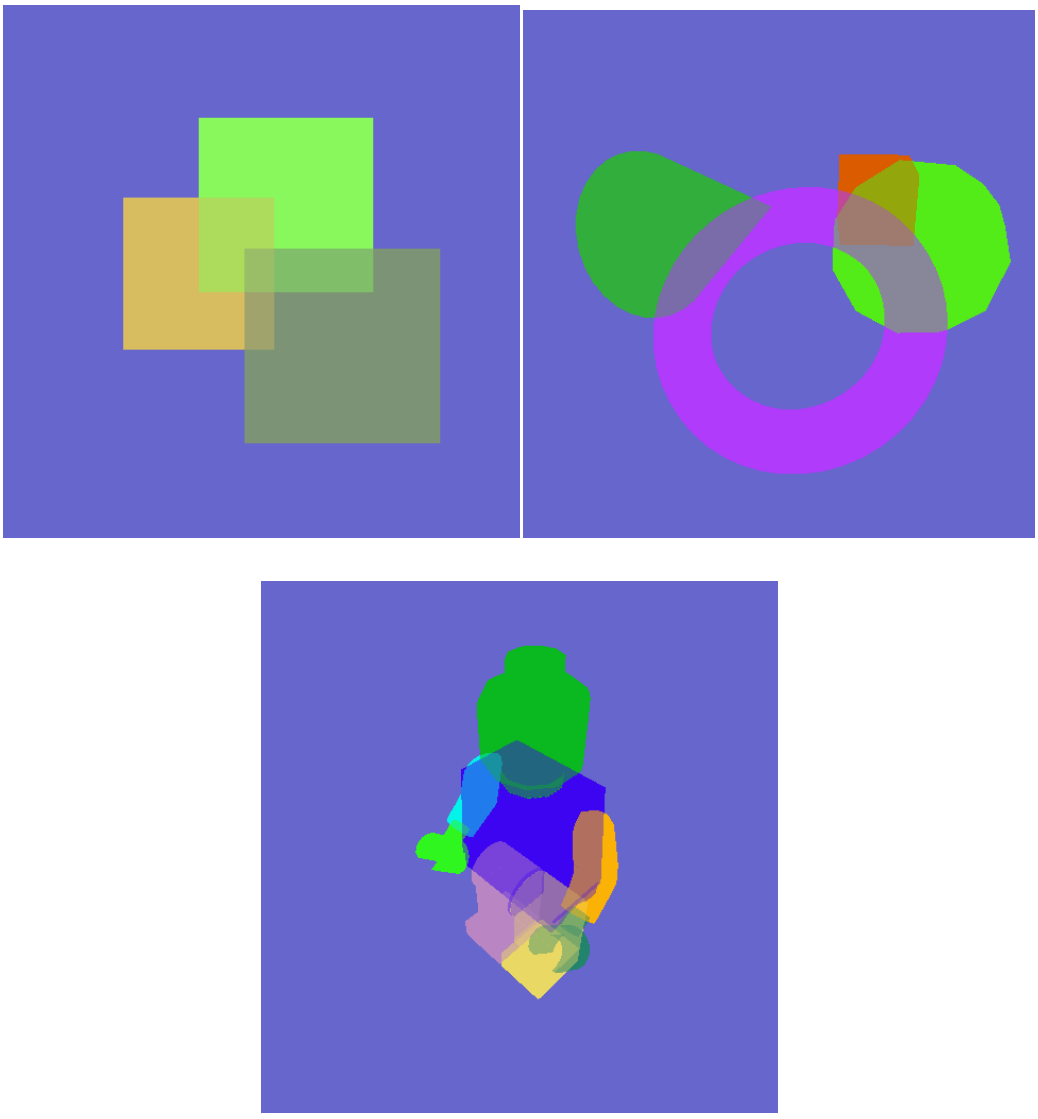


**Figura 3-9**

## 3.3.6 OPACITAT CREIXENT PER CAPA (PER OBJECTE)

### 3.3.6.1 DESCRIPCIÓ

La mateixa idea plantejada en el conjunt de tècniques d'opacitat per capes pot ser utilitzada també en aquest conjunt de solucions. Aquesta tècnica intenta donar més importància als objectes que es troben tapats per altres mentre que es perd informació espaial de la localització dels objectes mantenint, tot i això, les millores que incorpora assignar una única opacitat per cada objecte.

La fórmula que utilitza aquesta tècnica per obtenir la opacitat també requereix la informació que calcula la tècnica de *color coding* per conèixer en quines capes es troba cada objecte i saber, donat un píxel, quins objectes es troben projectats en ell i en quin ordre estan. Els elements que es necessiten per calcular la opacitat són el nombre de capes **n** en els quals està l'objecte representat i la mediana **m** de les capes en les quals es troba l'objecte. En resum, el color que s'utilitza per calcular el resultat de l'aplicació de la tècnica és el que dóna l'algorisme de *depth peeling* mentre que el color de les imatges que ofereix el *color coding* només serveix per a calcular els identificadors dels objectes.

La opacitat es calcula a partir de la següent fórmula: $m/n$. Un cop més, el color de fons no contribueix mai en el color final d'un píxel si en aquest hi ha projectat algun objecte, ja que si es donés al color de fons el comportament d'un objecte, en les escenes amb molta profunditat i poques capes ocupades amb objectes, la contribució del color de fons seria massa gran.

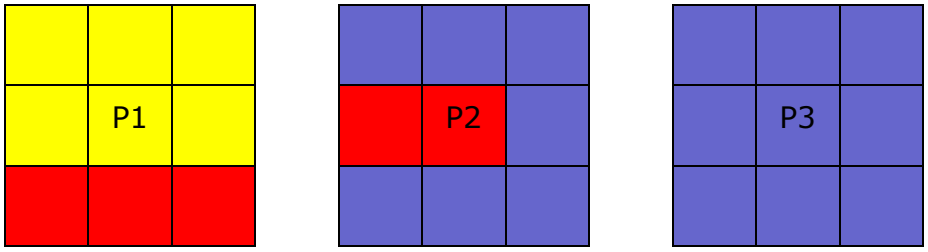
#### 3.3.6.2 IMPLEMENTACIÓ

Un aspecte que a tenir en compte són els resultats incorrectes que pot donar l'aplicació de la fórmula amb uns paràmetres concrets. En cas que la mediana de les capes tingui com a resultat 0, s'ha decidit per assignar la opacitat igual a 0.1. Si no es realitzés aquest canvi, es podria donar el cas de que la contribució al color final d'un objecte fos 0 i per tant, no es veiés.

Un cop es tenen en compte aquests detalls, els passos que segueix l'algorisme són els següents:

1. A partir del resultat d'aplicar l'algorisme de *color coding*, identificar per a cada objecte en quines capes es troba representat.
2. Per a cada objecte de l'escena, calcular la mediana **m** de les capes en les quals es troba. Aquesta és la capa en la que queda assignat l'objecte.
3. Per a cada píxel de la imatge:
    a. Inicialitzar el color de sortida S amb el codi RGB igual a (0,0,0).
    b. Accedir a la capa més propera a l'usuari.
    c. Comprovar que el color del píxel sigui diferent del color de fons. Si ho és, obtenir l'identificador de l'objecte que es troba projectat en el píxel. Altrament, finalitzar la iteració.
    d. Calcular la opacitat de l'objecte mitjançant la fórmula següent:
    $$Opacitat = m/n$$
    e. Multiplicar la opacitat pel color que es troba en el píxel.
    f. Sumar el resultat al color de sortida S.
    g. Repetir els passos **b** a **f** utilitzant cada cop una capa més llunyana a l'usuari.
    h. Normalitzar el color S dividint cada component del color per la suma d'opacitats utilitzades. En el cas de que S no hagi estat modificat, emmagatzemar en S el color de fons.
    i. Emmagatzemar el color de S en el píxel corresponent de la imatge de sortida.

### 3.3.6.3 EXEMPLES




En aquest exemple es poden veure 3 capes obtingudes a través de la tècnica de *depth peeling*. Per a veure el funcionament de l'algorisme, només s'utilitzaran els colors del píxel P1, que correspon a l'objecte 0 amb color RGB (255,255,0), el píxel P2, que pertany a l'objecte 2 i que té el color amb codi (255,0,0) i el píxel P3 que és color de fons amb codi (102,102,204). A continuació es mostren els passos que es segueixen:

- Per a cada objecte s'obté en quines capes està representat utilitzant el *color coding*.
  - Objecte 0: Capa 0.
  - Objecte 1: Capa 0, Capa 1.
- Per a cada objecte es calcula la mediana de les capes en les qual apareix.
  - Objecte 0: Mediana = 0
  - Objecte 1: Mediana = 0.5
- S'inicialitza el color de sortida S amb codi RGB (0,0,0).
- Iteració 1.
  - Es comprova que P1 és diferent del color de fons. Com ho és, s'obté que el píxel P1 correspon a l'objecte 0, que es troba representat en 1 capa i la seva mediana es 0, per tant la seva opacitat és de 0 / 1 = 0. Com que la seva opacitat no pot ser de 0, se li assigna una opacitat mínima de 0.1.
  - Es multiplica el color P1 per la opacitat calculada:
    - Component vermell = 255 * 0.1 = 25
    - Component verd = 255 * 0.1 = 25

        - Component blau = 0 * 0.1 = 0

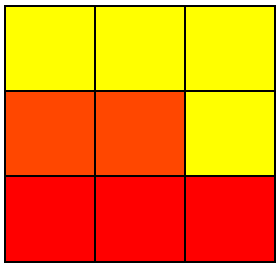
      Es suma el color calculat a S, el valor del qual passa a ser (25,25,0).

- Iteració 2.
    - Com que el color de P2 és diferent del color de fons, es calcula que el píxel P2 correspon a l'objecte 2, que es troba representat en 2 capes i la mediana de les capes on es troba és de 0.5. Per tant, se li assigna la opacitat de 0.5 / 2 = 0.25.
    - Es multiplica el color de P2 per la opacitat calculada:
        - Component vermell = 255 * 0.25 = 64
        - Component verd = 0 * 0.25 = 0
        - Component blau = 0 * 0.25 = 0

      Sumant aquests valors a S, aquest obté el nou valor de (89,25,0).
- Iteració 3.
    - Com que el color de P3 és igual al color de fons, la iteració finalitza.
- Es normalitza el color de S dividint per la suma d'opacitats utilitzades (0.1 + 0.25 = 0.35).
    - Component vermell = 89 / 0.35 = 255
    - Component verd = 25 / 0.35 = 71
    - Component blau= 0 / 0.35 = 0

  Per tant, el color que s'emmagatzema en la imatge de sortida és (255,71,0).

En la següent imatge es pot veure el resultat després d'aplicar l'algorisme per a tots els píxels de la imatge:



A continuació, en la figura 3-10, es mostra el resultat d'aplicar l'algorisme en escenes de major complexitat amb una profunditat de 15 capes.

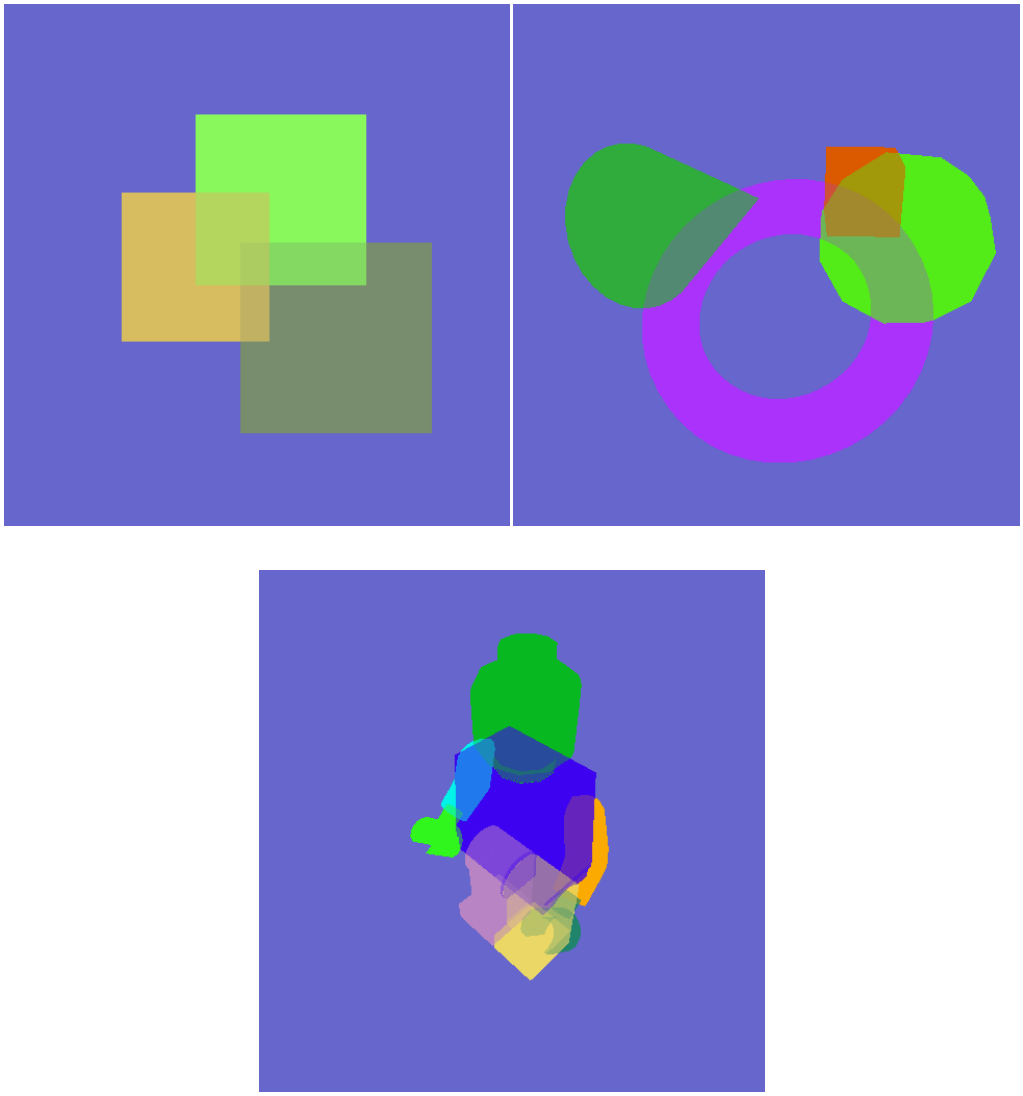


**Figura 3-10**

### 3.3.7 OPTIMITZACIÓ D'OPACITAT PER OBJECTE

#### 3.3.7.1 DESCRIPCIÓ

Enfocant de manera totalment diferent a la resta de tècniques s'ha decidit crear un algorisme voraç que generi una imatge seleccionant unes opacitats pels objectes de manera més intel·ligent que la resta de solucions. Un algorisme voraç o en anglès *greedy*, consisteix en la utilització d'un heurístic que escull una solució òptima en cada pas local intentant arribar a una solució òptima general.

En el cas particular que tracta aquest apartat l'algorisme funciona de la següent forma:

1. S'inicialitza una imatge de la mida desitjada.
2. Es pinta l'objecte més llunyà varies vegades amb una opacitat diferent i s'emmagatzema el resultat en diferents imatges composant el color obtingut amb el color de la imatge inicialitzada (si es la primera iteració) o la imatge obtinguda en la iteració anterior. Aquestes imatges són les solucions locals de la actual iteració.
3. L'heurístic (veure més endavant) escull quina imatge té el major nombre d'objectes identificables, que és la solució local òptima actual, i s'emmagatzema per a ser utilitzada en la utilització de la següent iteració.
4. Es repeteixen els passos 2-4 utilitzant cada cop un objecte més proper.

Com es pot veure en la descripció, ja no s'utilitza el resultat d'aplicar la tècnica de *depth peeling* sinó que tan sols s'utilitzen les eines que ofereix *OpenGL*, donat que amb elles es pot realitzar tota la feina necessària.

A la figura 3-11 es mostren 9 imatges en les que es pot veure el conjunt de solucions locals on l'objecte que es tracta és un torus amb les seves diferents opacitats. La solució local òptima escollida és la imatge amb el requadre vermell.

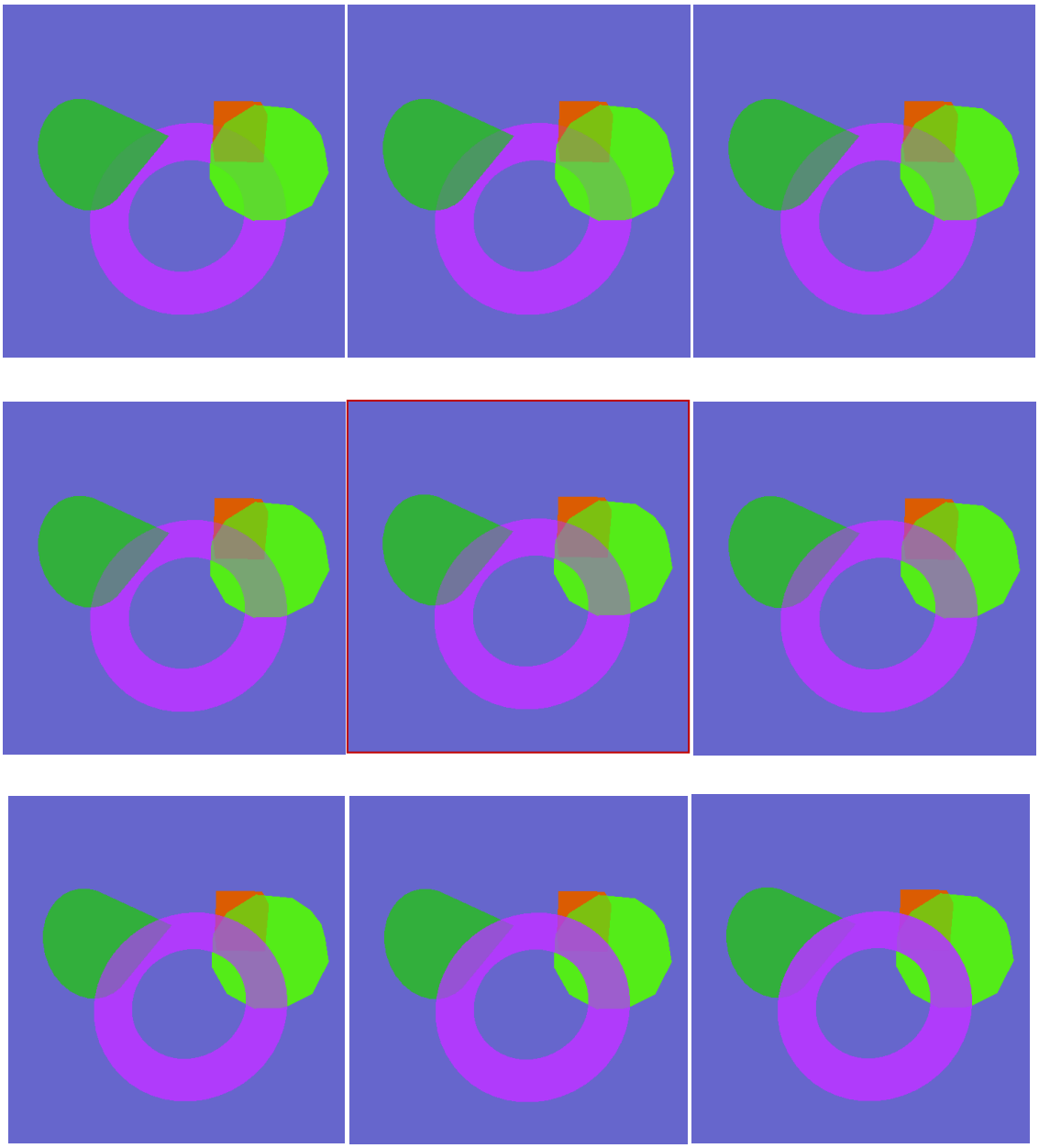


**Figura 3-11**

### 3.3.7.2 IMPLEMENTACIÓ

La implementació de l'algorisme ha resultat ser la més complexa de totes les tècniques ja que per al correcte funcionament de l'algorisme descrit a l'apartat 4.2.7.1 cal realitzar una sèrie de tasques que no són trivials. Obtenir la llista d'objectes ordenats de més llunyà a més proper a la càmera

no és immediat i els passos que s'han seguit per complir aquest objectiu han sigut els següents:

1. Calcular el punt mig de cada objecte a partir de les coordenades dels vèrtexs dels objectes. Realitzar l'ordenació utilitzant només un punt per objecte enlloc de per primitiva accelera de manera molt significativa els càlculs.
2. Com que el que es vol és una ordenació de distància respecte la càmera (o usuari) es converteixen els punts calculats dels objectes que es troben a l'espai de coordenades d'objecte, en coordenades de dispositiu.
3. La distància dels objectes fins a la càmera està representada mitjançant la coordenada z dels punts projectats, essent les z menors les corresponents als objectes mes properes a la càmera i les majors les dels objectes més llunyans. Per tant, s'ordenen les coordenades z utilitzant l'algorisme de *quick sort*.

Les opacitats que s'apliquen als objectes per generar el conjunt de solucions locals estan en el rang 0.1 a 0.9 ambdós inclosos. Per a composar el color de les imatges resultants al afegir a la imatge de la iteració anterior el nou objecte amb la seva opacitat, s'ha optat per seguir la mateixa fórmula que utilitza *alpha blending*, en la que per a cada píxel de la imatge es calcula el color de la següent forma, si $C_A$ és el color que ja hi ha a la imatge i $C_N$ el color del nou objecte:

$$C = C_N \cdot \alpha + (1 - \alpha) \cdot C_A$$

L'heurístic utilitzat no s'explica en aquest apartat, però es pot trobar tota la informació del seu funcionament a l'apartat 4.3, on es descriu com decidir quina imatge mostra més objectes i quines millores s'han realitzat per a optimitzar-lo.

#### 3.3.7.2.1 MILLORES

A partir de l'algorisme explicat s'ha decidit fer unes petites millores, o aportacions, que ofereixen un resultat que intenta explotar les condicions que utilitza l'heurístic per decidir quina imatge ofereix una visualització major d'objectes.

Com que l'heurístic treballa a partir dels contorns dels objectes decidint que les imatges que mostren un nombre major de contorns ben definits són solucions òptimes, s'ha optat per pintar els contorns *externs* dels objectes de color negre, maximitzant d'aquesta manera, el nombre de contorns ben definits.

Per a obtenir el contorn dels objectes es podria haver optat per aplicar un filtre detector d'arestes, però s'ha decidit implementar un sistema senzill que generi aquests contorns perquè d'aquesta manera s'obté un major control. L'algorisme que dibuixa els contorns funciona de la següent manera:

1. Generar una imatge S amb tots els píxels de color blanc, amb codi RGB (255,255,255).
2. Pintar l'objecte més llunyà a la càmera i emmagatzemar el resultat en una imatge.
3. Per a cada píxel de la imatge:
    a. Comprovar que el color del píxel és diferent del color de fons.
        i. Si ho és, es miren els 8 veïns del píxel i en el cas de que algun dels veïns del píxel sigui color de fons, es decideix que el píxel és del contorn i per tant, es pinta en el píxel corresponent de la imatge S de color negre.
        ii. En el cas de que el píxel sigui de color de fons, es passa al següent píxel.
4. Repetir els passos 2-3 amb un objecte cada cop més proper a la càmera.

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | P | 5 |
| 6 | 7 | 8 |

**Figura 3-12. Els 8 veïns del píxel P.**

Un cop es té la imatge amb el contorn i la imatge resultant d'aplicar l'algorisme voraç, es combinen de la següent forma:

1. Per a cada píxel de la imatge:
    a. Si el píxel corresponent en la imatge del contorn és de color negre, al píxel de la imatge de sortida es pinta de color negre. Altrament, es pinta del color del píxel de la imatge de l'algorisme voraç.

En la figura 3-13 es mostra un exemple d'un model senzill del qual s'han calculat els contorns dels objectes.

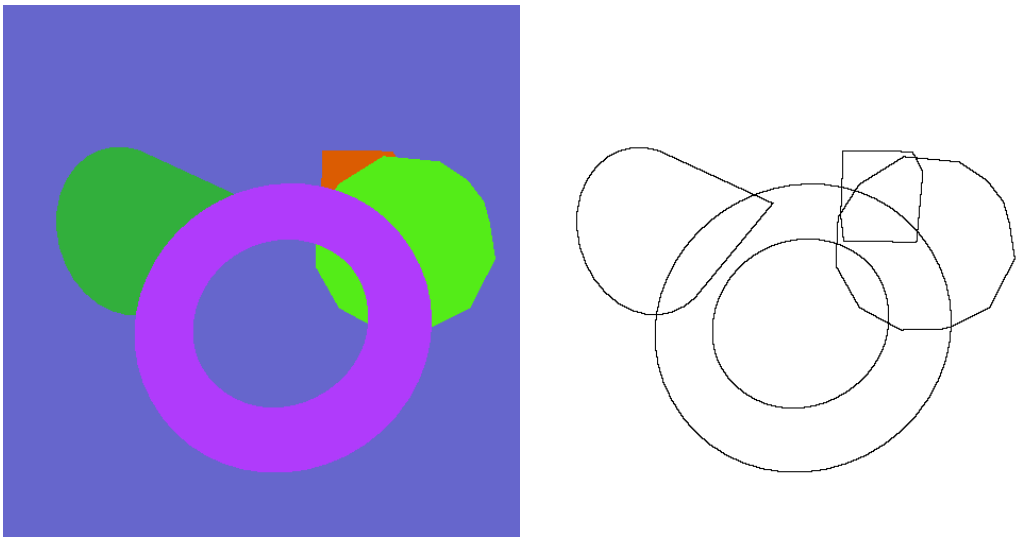


**Figura 3-13**

### 3.3.7.3 EXEMPLES

En la figura 3-14 es mostra el resultat d'aplicar la tècnica en una escena i al seu costat, el resultat de la millora.

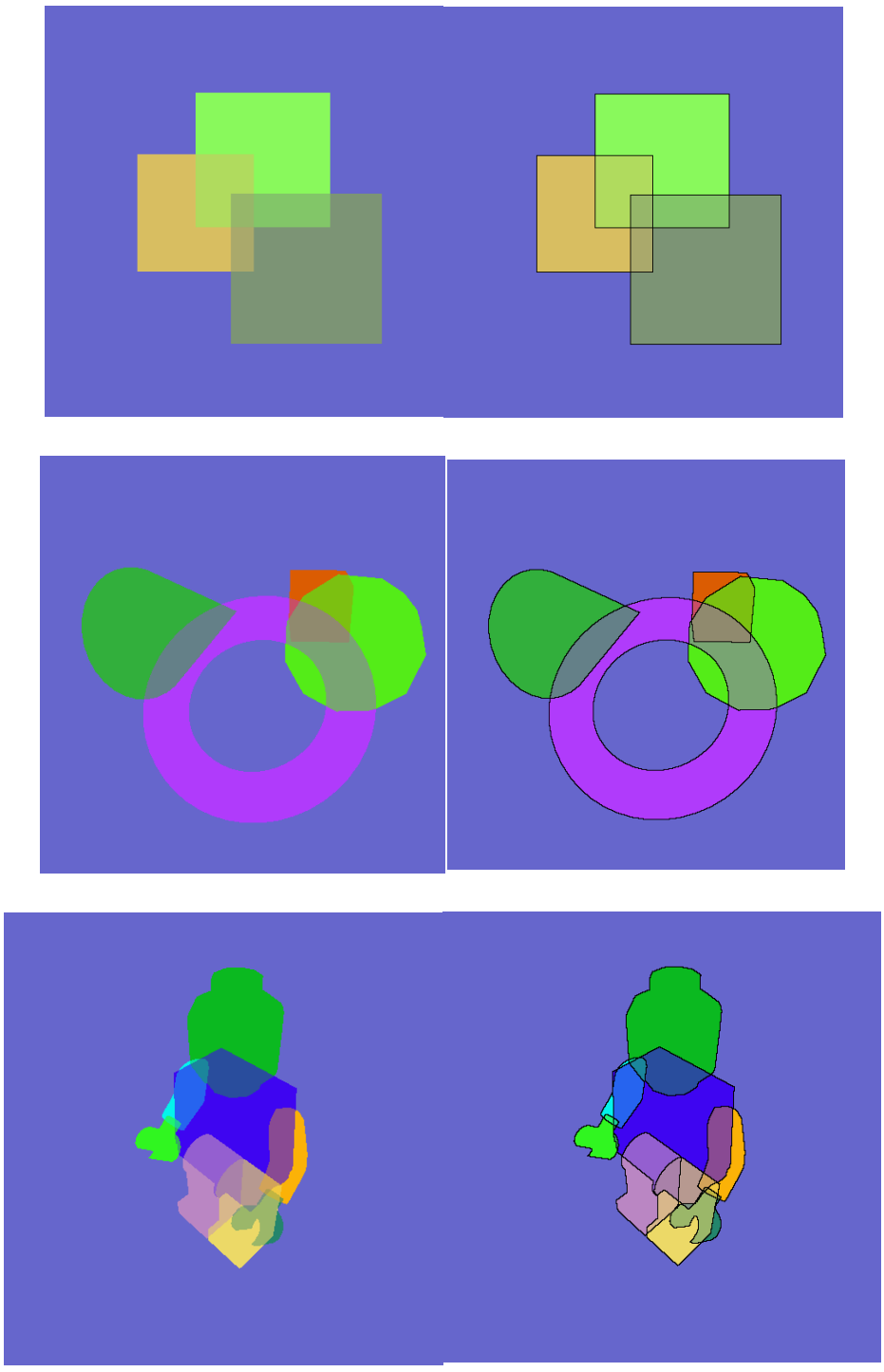


**Figura 3-14**

## 3.4 L'HEURÍSTIC

### 3.4.1 DESCRIPCIÓ

L'heurístic utilitzat en la tècnica d'optimització d'opacitat per objecte calcula el nombre de píxels ben contrastats que hi ha a la imatge i escull la imatge que té el major nombre. Un píxel ben contrastat és aquell que el seu color està ben diferenciat dels colors dels seus píxels veïns. Els valors detallats que utilitza per definir què és el terme "ben diferenciat" s'explica en el proper apartat.

A part de la seva funció per a la implementació de la tècnica d'optimització d'opacitat per objecte, l'heurístic també s'ha utilitzat per a comparar les diferents imatges obtingudes al aplicar les tècniques utilitzades obtenint d'aquesta manera uns valors numèrics que permeten, en teoria, conèixer quines solucions són millors en l'aspecte de nombre d'objectes identificables.

### 3.4.2 DISSENY I IMPLEMENTACIÓ

L'apartat més importat de l'heurístic és el sistema que utilitza per a determinar si un píxel està ben definit o no. Per a obtenir un resultat òptim, s'han realitzat vàries proves dibuixant varies formes amb increments de color cada cop majors i sobre aquest dibuixos s'ha aplicat l'algorisme detector d'arestes de Sobel. Amb aquest resultat s'ha trobat quina ha de ser la diferencia mínima de color entre 2 píxels per tal que l'algorisme de Sobel detecti que existeix una aresta ben definida. Aquesta diferència, que ha estat utilitzada per l'heurístic ha de ser de la següent forma, tenint en compte les components r,g,b prenen valors entre 0 i 255:

$$(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2 \geq 1200$$

Per a la comparació d'un píxel amb els seus veïns per saber si està ben definit o no, s'ha escollit agafar la mitja de la diferència de color entre el píxel i el seu veí superior i el píxel amb el seu veí de la dreta. Si la mitja és superior al valor de 1200, llavors es decideix que està ben definit.

En definitiva, l'algorisme queda de la següent forma:

1. Inicialitzar una variable S a 0 que emmagatzema el número de píxels ben definits.
2. Per a cada píxel de la imatge:
    a. Calcular la diferència de color amb el píxel de la dreta.
    b. Calcular la diferència de color amb el píxel que es troba en la fila superior i en la mateixa columna.
    c. Calcular la mitja de les diferències calculades. En el cas de ser superior a 1200, incrementar S.

#### 3.4.2.1 MILLORES

A l'algorisme descrit s'han afegit alguns canvis que intenten millorar els resultats obtinguts amb l'heurístic originat. El primer canvi consisteix en no donar la mateixa importància a tots els píxels que resulten ser ben definits. Aquesta millora permet que objectes petits que mitjançant l'algorisme tal i com s'ha definit anteriorment apareixerien tapats per grans objectes amb un gran contorn ben definit, apareguin visibles a l'escena. D'aquesta manera s'obtindrà un resultat més proper al desitjat ja que es busca maximitzar el nombre d'objectes visibles a l'escena.

Amb aquesta finalitat, per tant, s'ha decidit que els píxels ben definits que no formen part del contorn tinguin un pes major que els píxels ben definits del contorn (en la implementació els píxels que no són del contorn valen dues vegades més que els que ho són) donant d'aquesta manera una importància superior als píxels dels objectes tapats.

Lògicament, aquesta millora és implementada només a l'heurístic que utilitza l'algorisme de la tècnica d'optimització d'opacitat per objecte quan un nou objecte es afegit a la imatge existent i es valora quina imatge amb el nou objecte ofereix un major nombre de píxels ben definits. A més, per a la implementació d'aquesta millora també s'utilitza la imatge que té el contorn del nou objecte per saber si el píxel que s'està mirant és part del contorn o no.

L'altra millora afegida a l'heurístic intenta optimitzar el temps de càlcul a l'hora de comparar les diferents imatges de les solucions locals de l'algorisme voraç. Les úniques diferències de resultat de l'heurístic en les imatges es troben en la zona on ha estat pintat el nou objecte, ja que la resta de la imatge és igual per totes les imatges, per tant, s'ha buscat un mètode per acotar la zona on aplicar l'heurístic.

El mètode més senzill que ha sorgit d'aquesta idea ha estat utilitzar la caixa contenidora dels l'objectes. Una caixa contenidora és la caixa mínima que conté tots els punts de l'objecte. Amb les coordenades dels punts de la capsa, que es troben en espai de coordenades d'objecte es transformen en coordenades d'espai de coordenades de dispositiu, utilitzant les matrius *modelview* i *projection* i obtenint com a resultat un hexàgon. Es calcula llavors el rectangle contenidor, que és el rectangle mínim que conté l'hexàgon, i per extensió, també conté l'objecte, tot i que no es el rectangle mínim contenidor de l'objecte en coordenades de dispositiu. Amb aquest resultat, s'obté una zona molt més reduïda on l'heurístic ha de comptar el nombre de píxels accelerant de manera significativa el temps de càlcul.

# 4 RESULTATS

## 4.1 DESCRIPCIÓ

Els models utilitzats per realitzar tant les proves d'usuaris com les mostres que s'ensenyen a continuació han estat escollits perquè mostren unes escenes amb un alt grau d'objectes tapats i a més, els objectes que contenen poden ser fàcilment reconeguts per qualsevol tipus d'usuari.

Per a crear els models s'ha utilitzat l'aplicació de *Sweet Home* 3D, que permet la creació de models d'oficines, cases, etc. de manera senzilla mitjançant *Drag&Drop* aconseguint així models amb una alta complexitat. A més, permet l'exportació dels models en format OBJ i agrupa els objectes en diferents *object groups,* permetent aprofitar d'aquesta manera al màxim l'aplicació creada.

En el segon apartat es mostren el conjunt de les imatges obtingudes a partir de les tècniques implementades, que són les 6 descrites més la millora de la tècnica d'optimització d'opacitat per objecte, mostrant-les en el mateix ordre en el qual han estat explicades. A continuació s'ensenyen zones particulars de les imatges on es poden veure en detall les diferències entre les tècniques en les que es pot veure amb més o menys dificultat diferents objectes.

Al tercer apartat s'ensenyen els resultats a l'aplicar l'heurístic en les imatges descobrint així quines imatges, en teoria, són més adients per a la identificació d'objectes.

Finalment, en l'últim apartat s'explica quins tests d'usuari s'han realitzat i es comenta els resultats obtinguts comparant-los amb els de l'heurístic i descobrint d'aquesta manera si l'heurístic ha estat implementat de manera correcta o no.

## 4.2 IMATGES

La primera escena consisteix en una planta d'un edifici d'oficines. En les tècniques que utilitzen *depth peeling*, s'han fet servir 25 capes de profunditat.

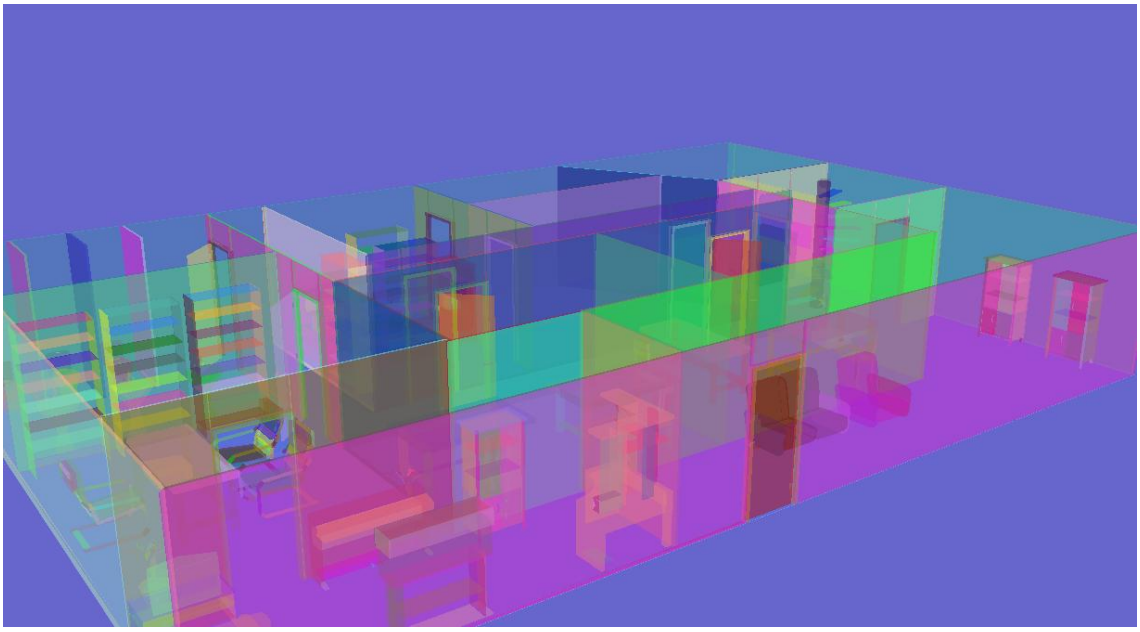


**Figura 4-1. Opacitat uniforme.**

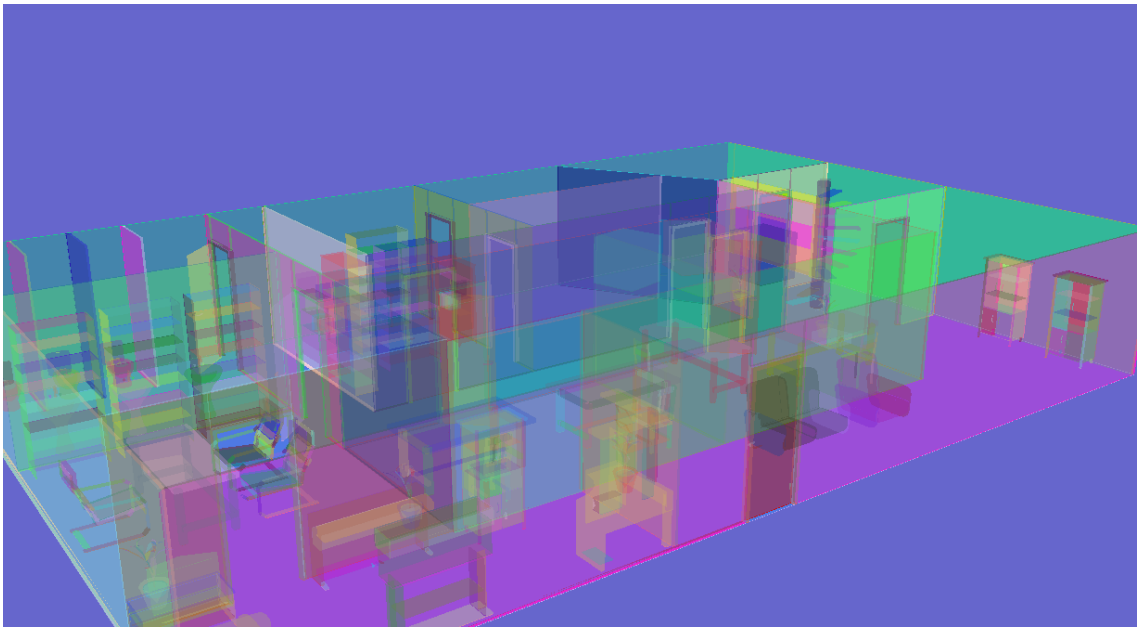


**Figura 4-2. Opacitat decreixent per capa (per píxel).**

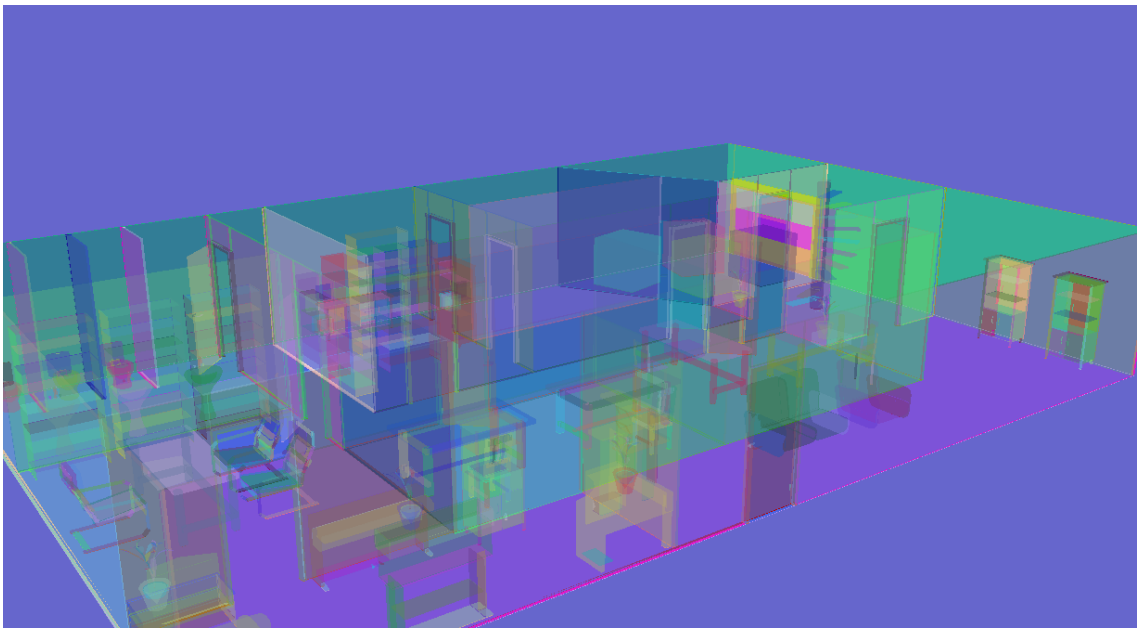


**Figura 4-3. Opacitat creixent per capa (per píxel).**

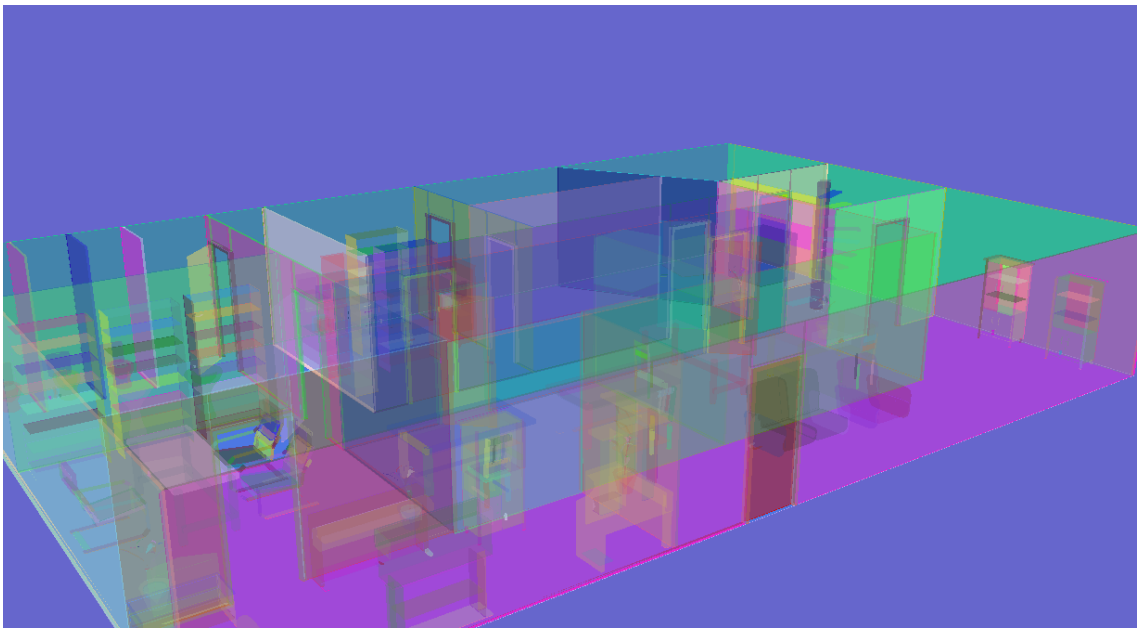


**Figura 4-4. Opacitat decreixent per capa (per objecte).**

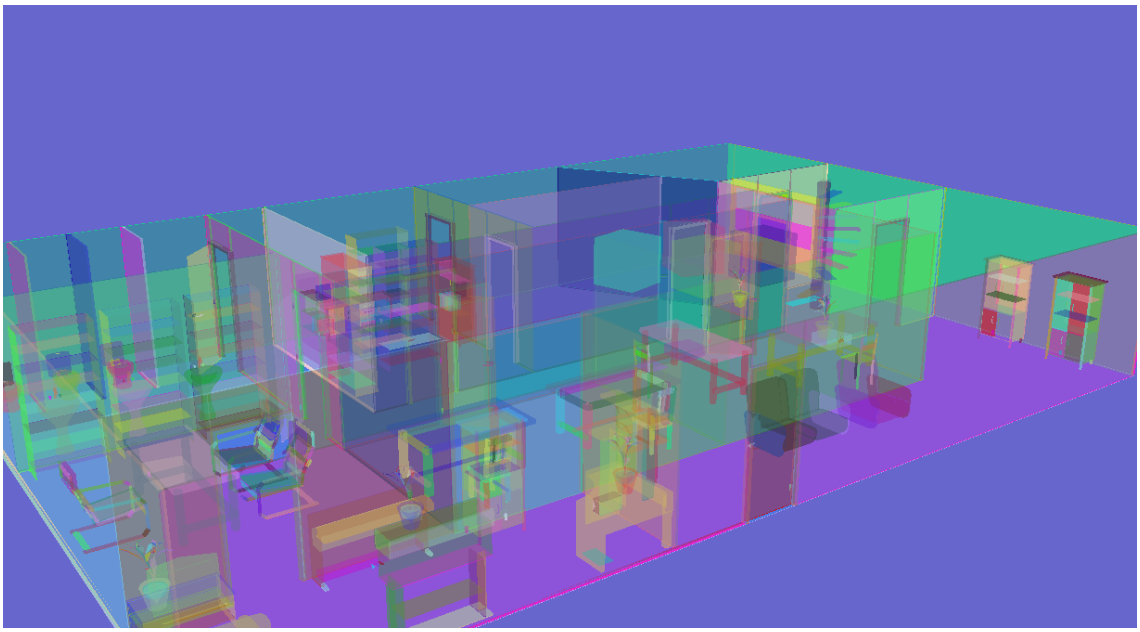


**Figura 4-5. Opacitat creixent per capa (per objecte).**

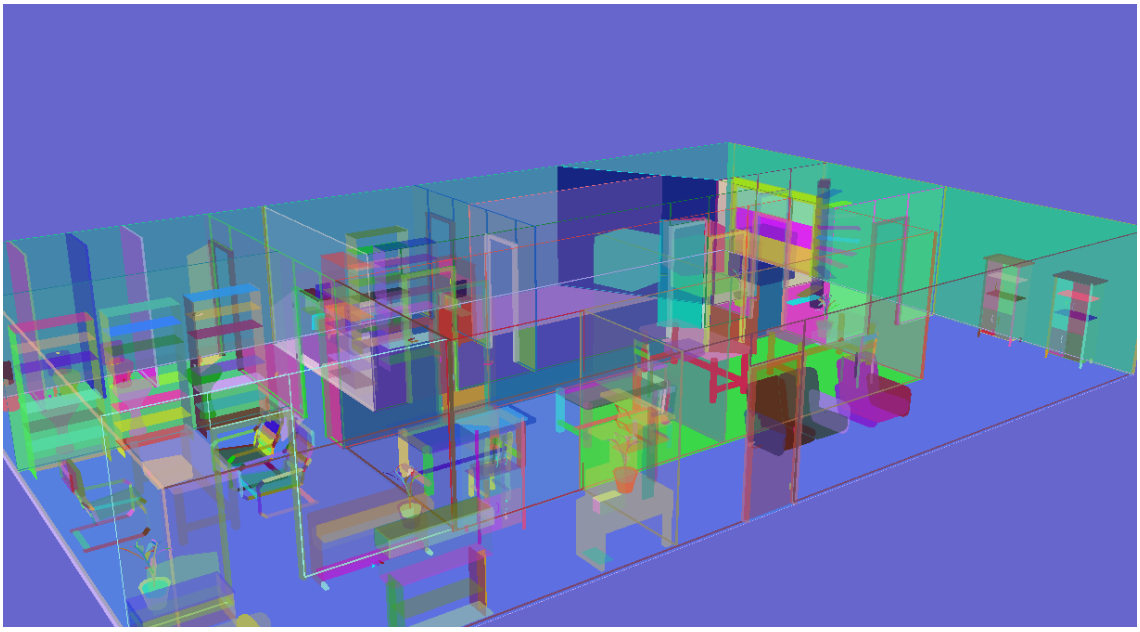


**Figura 4-6. Optimització d'opacitat per objecte.**

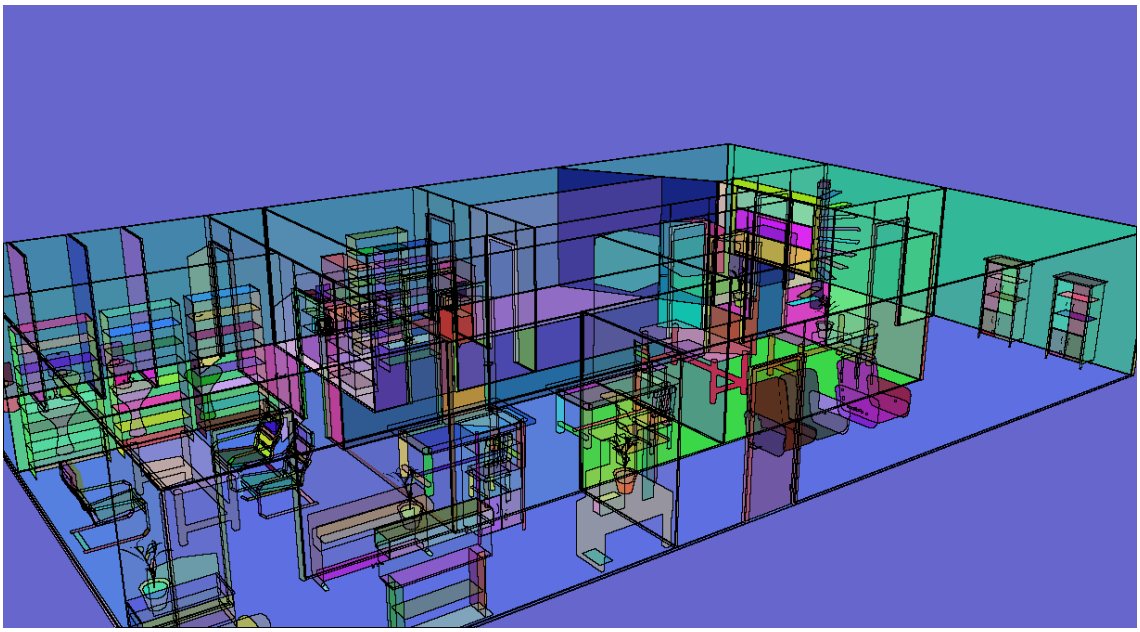


**Figura 4-7. Optimització d'opacitat per objecte amb contorn.**

En el proper grup d'imatges, es mostra amb detall la zona dels lavabos de la oficina. L'ordre que es segueixen les imatges és el mateix que el de lectura.

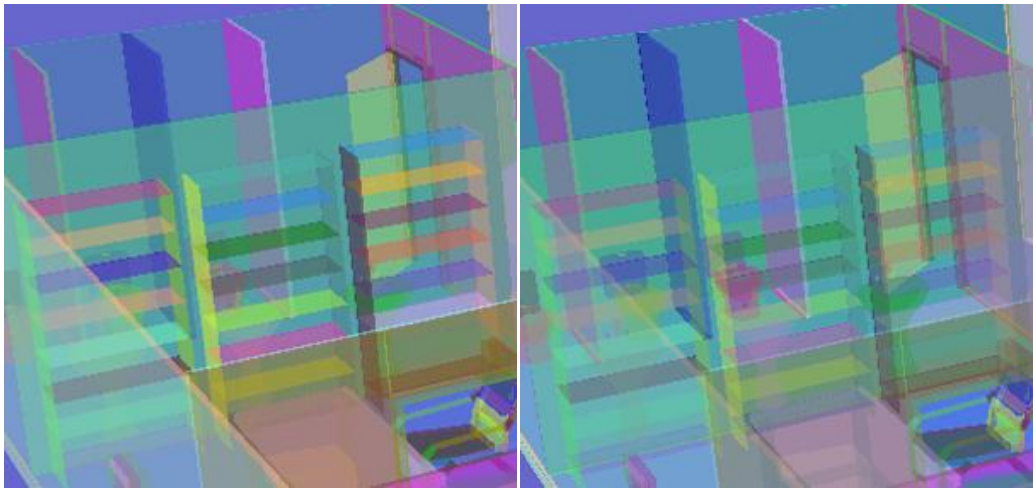


**Figura 4-8. Opacitat uniforme i opacitat decreixent per capa (per píxel).**

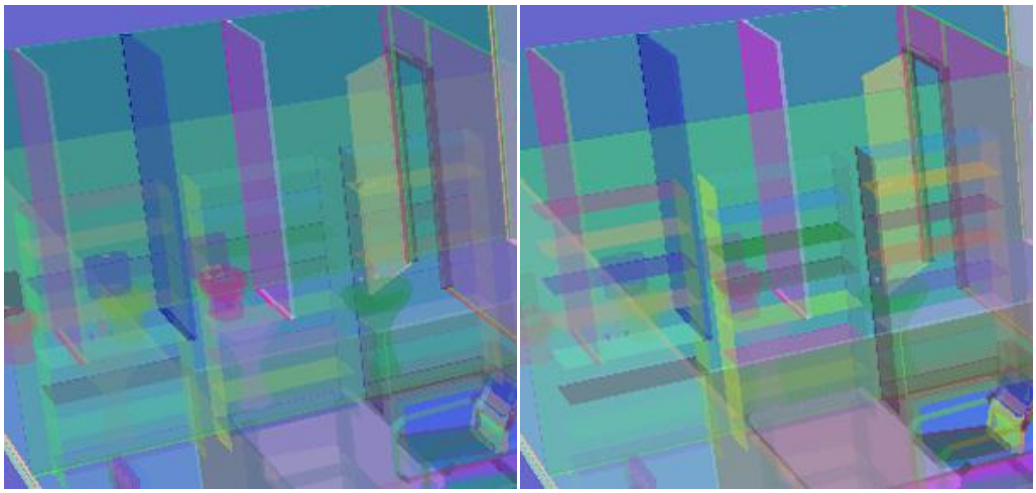


**Figura 4-9. Opacitat creixent per capa (per píxel) i opacitat decreixent per capa (per objecte).**

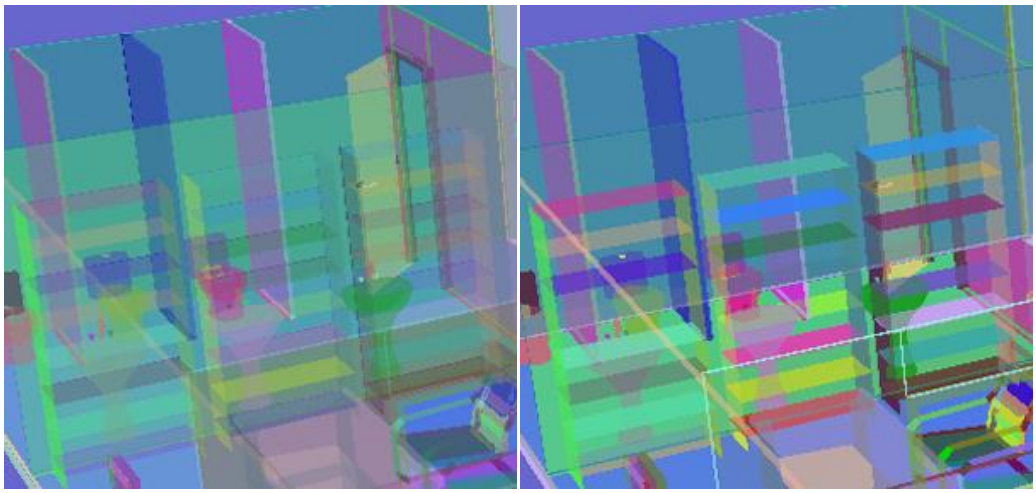


**Figura 4-10. Opacitat creixent per capa (per objecte) i optimització d'opacitat per objecte.**

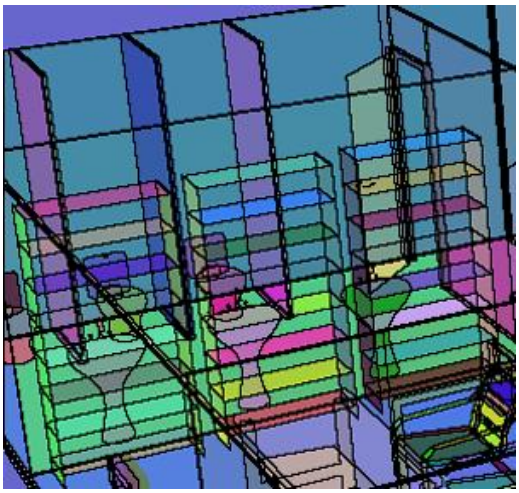


**Figura 4-11. Optimització d'opacitat per objecte amb contorn.**

En les següents imatges es centra l'atenció sobre la sala que té les escales cap al pis superior, que té 2 plantes i 3 portes.

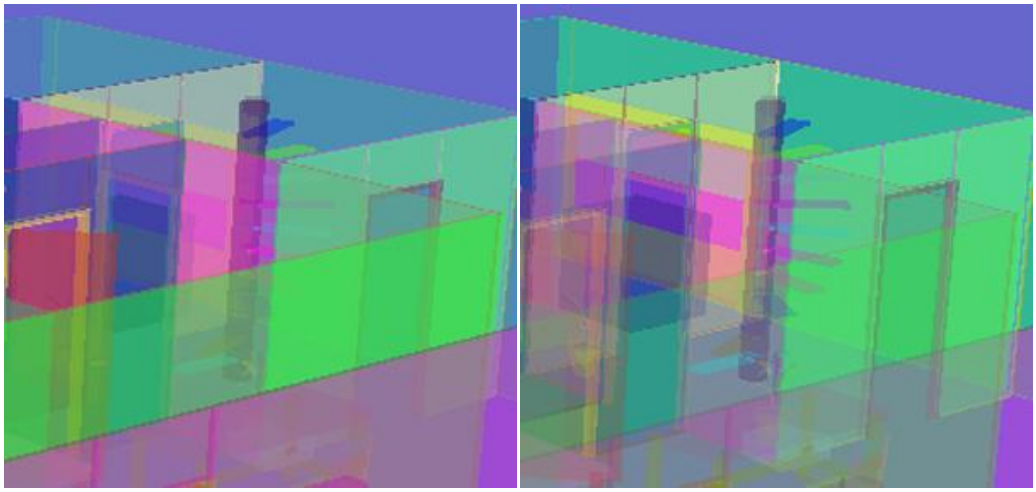


**Figura 4-12. Opacitat uniforme i opacitat decreixent per capa (per píxel).**

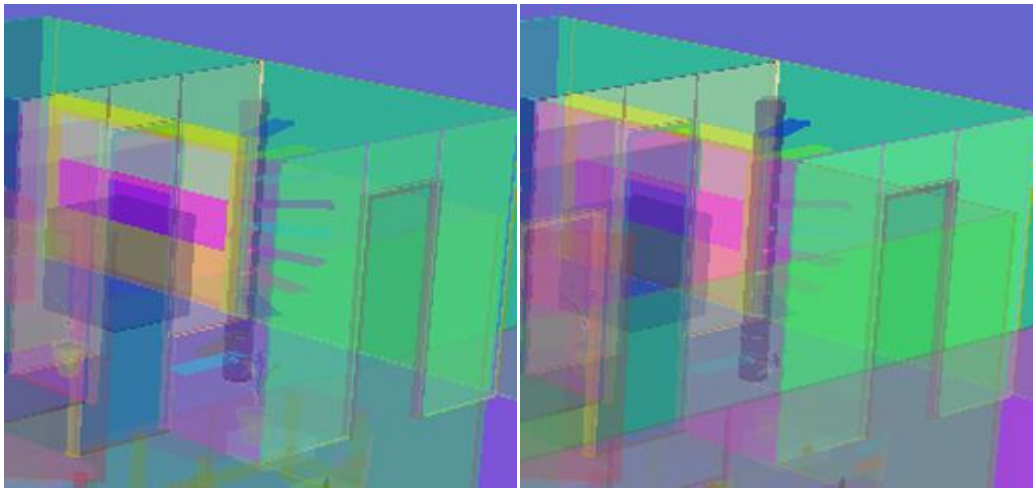


**Figura 4-13. Opacitat creixent per capa (per píxel) i opacitat decreixent per capa (per objecte).**

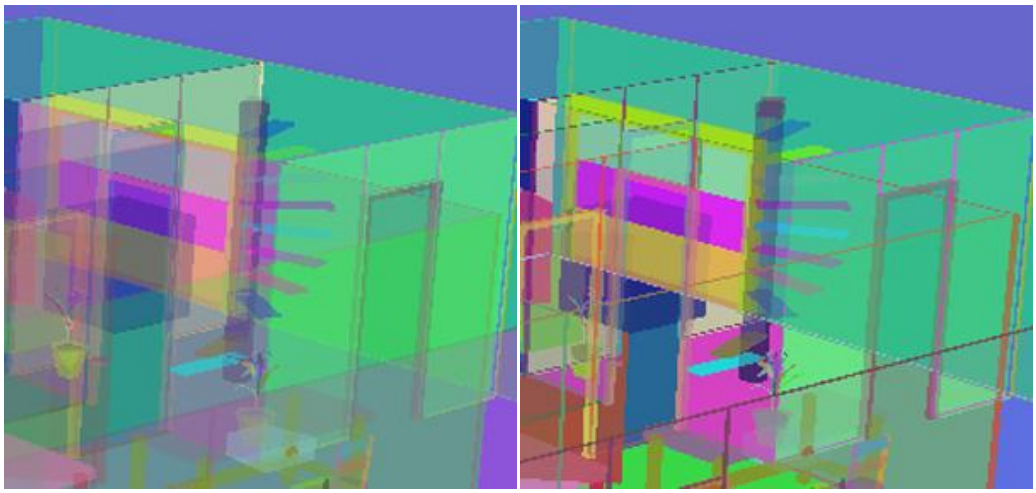


**Figura 4-14. Opacitat creixent per capa (per objecte) i optimització d'opacitat per objecte.**

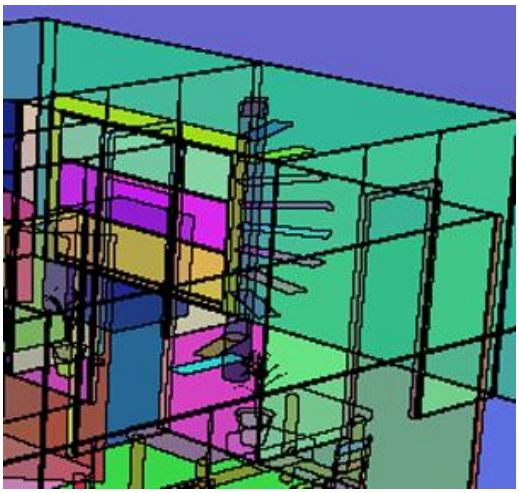


**Figura 4-15. Optimització d'opacitat per objecte amb contorn.**

El segon model correspon a una petita casa dividida en dues plantes i poques sales, 3 en la planta superior i 4 en la planta inferior. La complexitat d'aquest model consisteix en que al tenir 2 nivells d'alçada dificulta saber a quina planta pertany cada objecte.

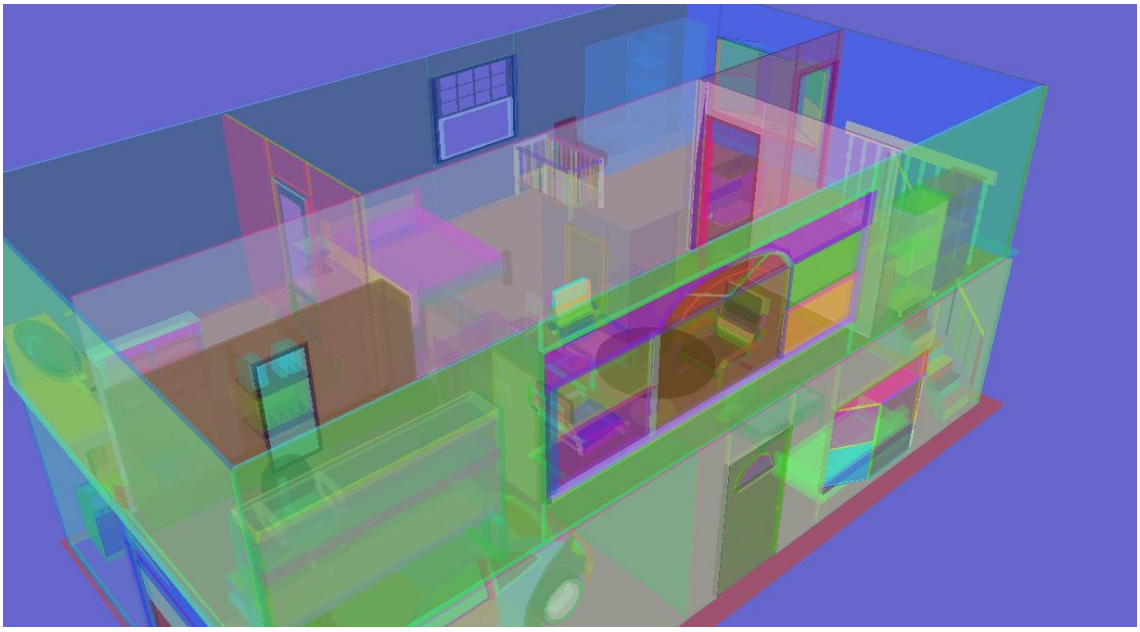


**Figura 4-16. Opacitat uniforme.**

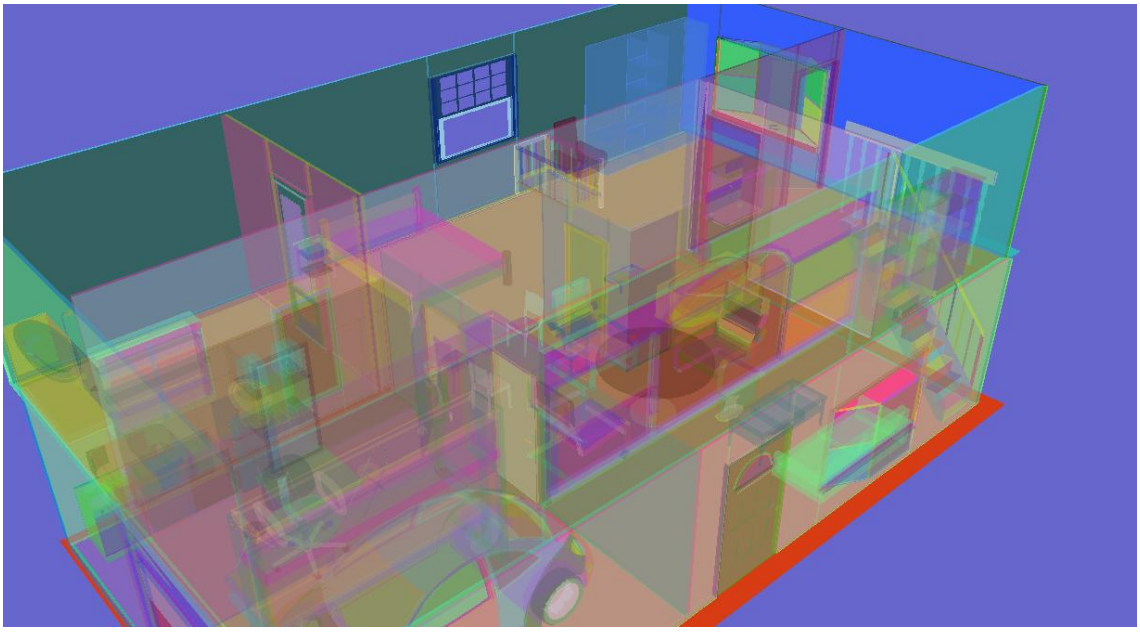


**Figura 4-17. Opacitat decreixent per capa (per píxel).**

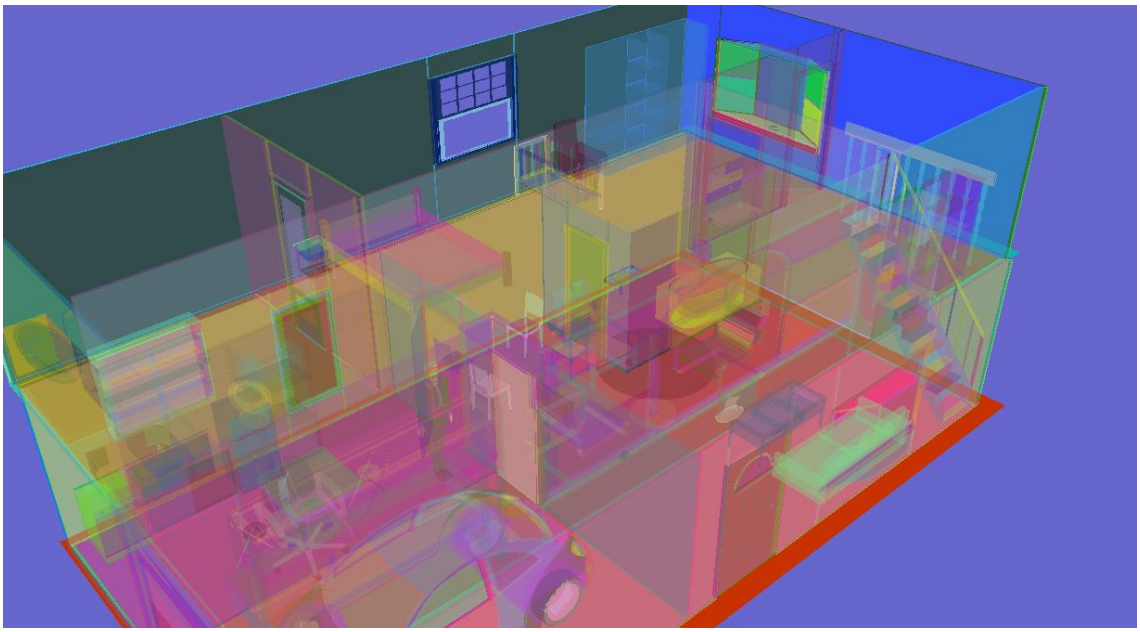


**Figura 4-18. Opacitat creixent per capa (per píxel).**

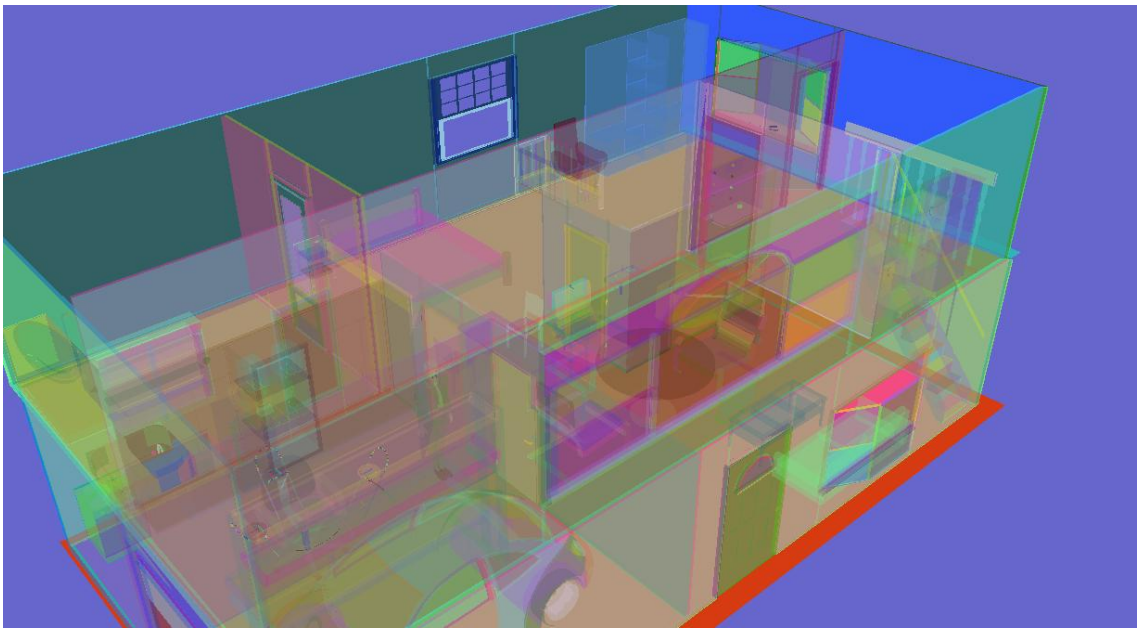


**Figura 4-19. Opacitat decreixent per capa (per objecte).**

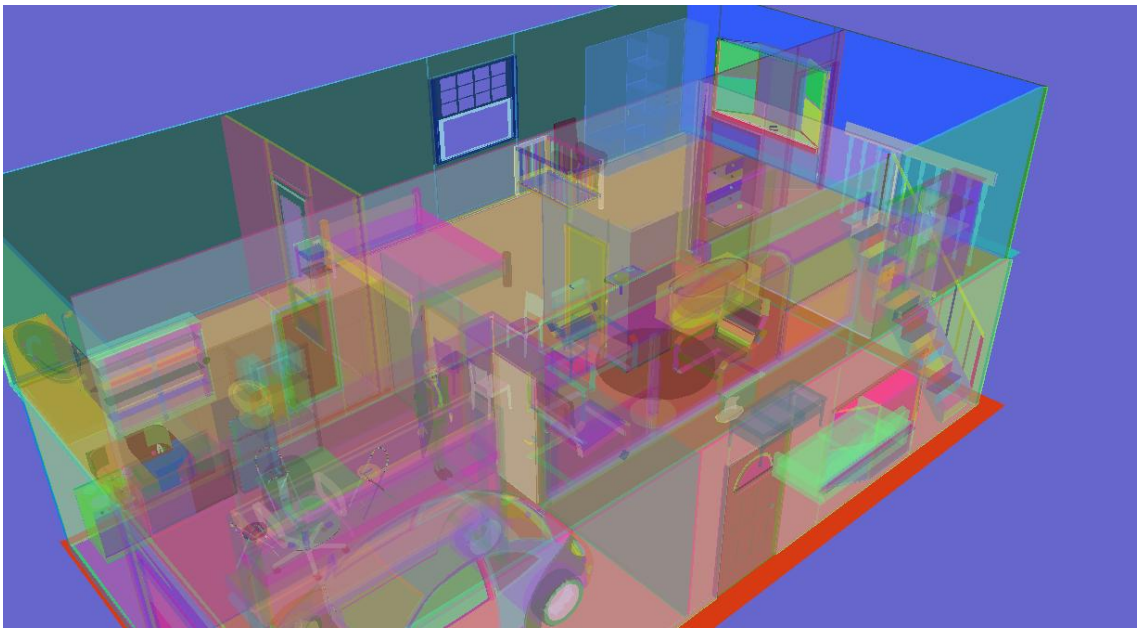


**Figura 4-20. Opacitat creixent per capa (per objecte).**

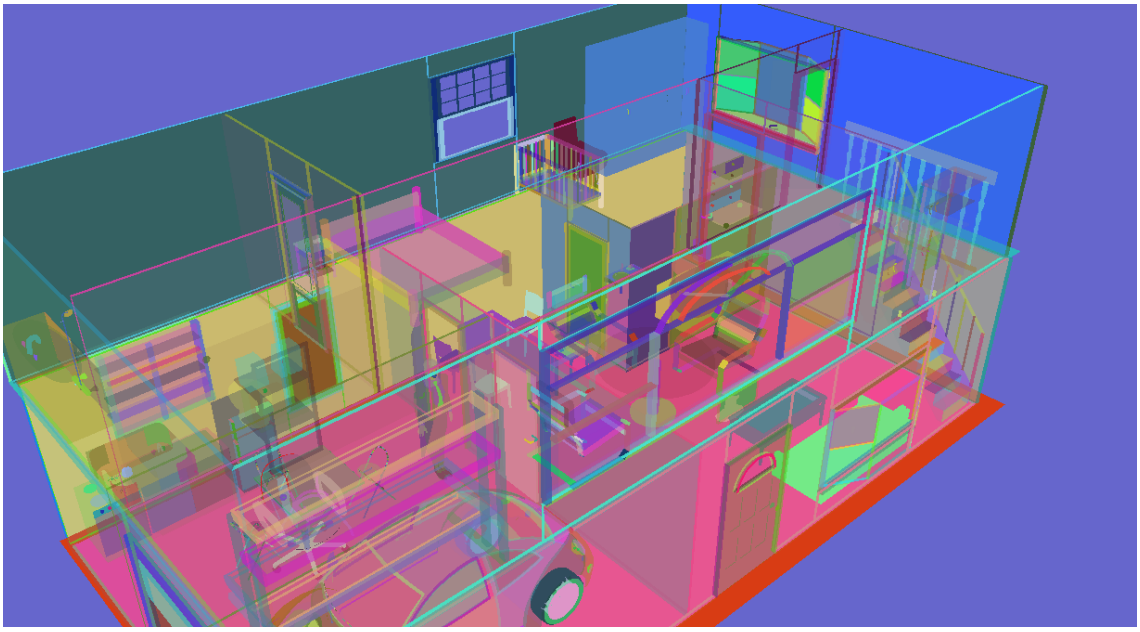


**Figura 4-21. Optimització d'opacitat per objecte.**

**Figura 4-22. Optimització d'opacitat per objecte amb contorn.**

En les següents imatges és mostra amb una mida major la zona on hi ha la taula del menjador, que es troba en la planta inferior.

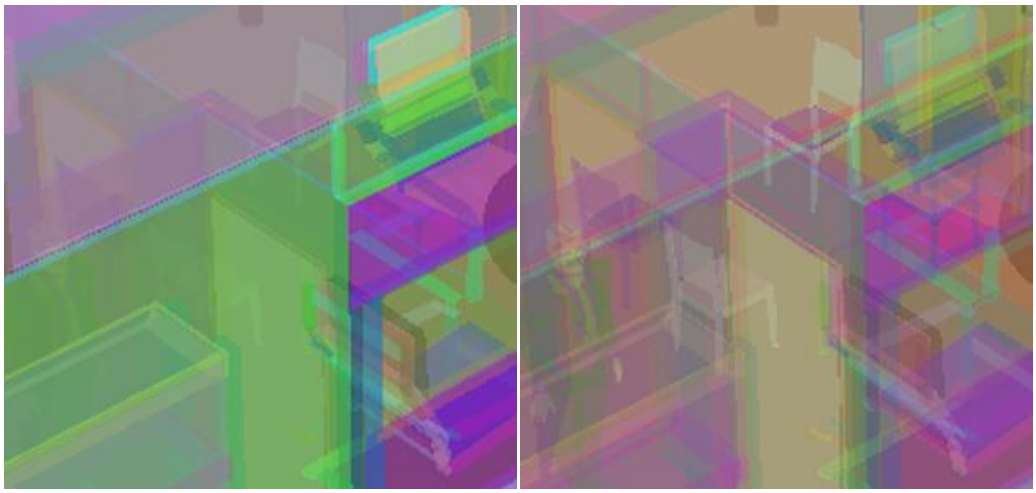


**Figura 4-23. Opacitat uniforme i opacitat decreixent per capa (per píxel).**

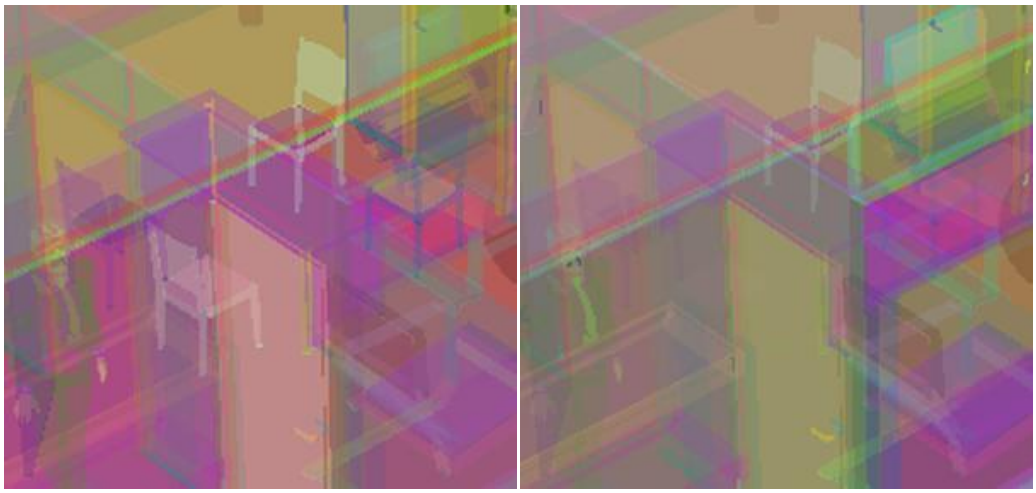


**Figura 4-24. Opacitat creixent per capa (per píxel) i opacitat decreixent per capa (per objecte).**

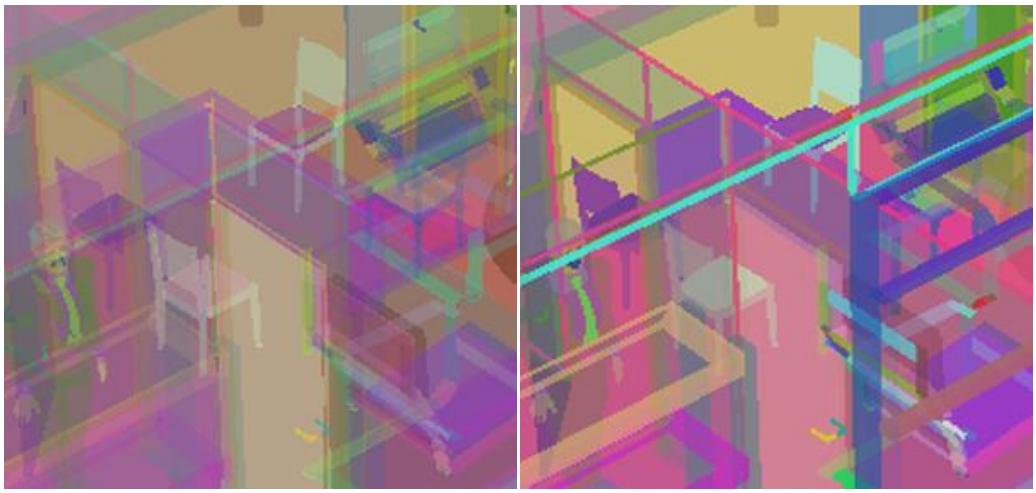


**Figura 4-25. Opacitat creixent per capa (per objecte) i optimització d'opacitat per objecte.**

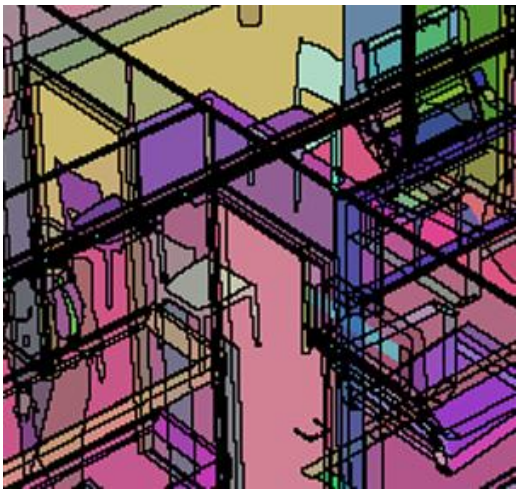


**Figura 4-26. Optimització d'opacitat per objecte amb contorn.**

Finalment, en les properes imatges es mostra en detall la zona de la segona planta on hi ha una taula rodona amb 3 cadires davant d'una finestra.

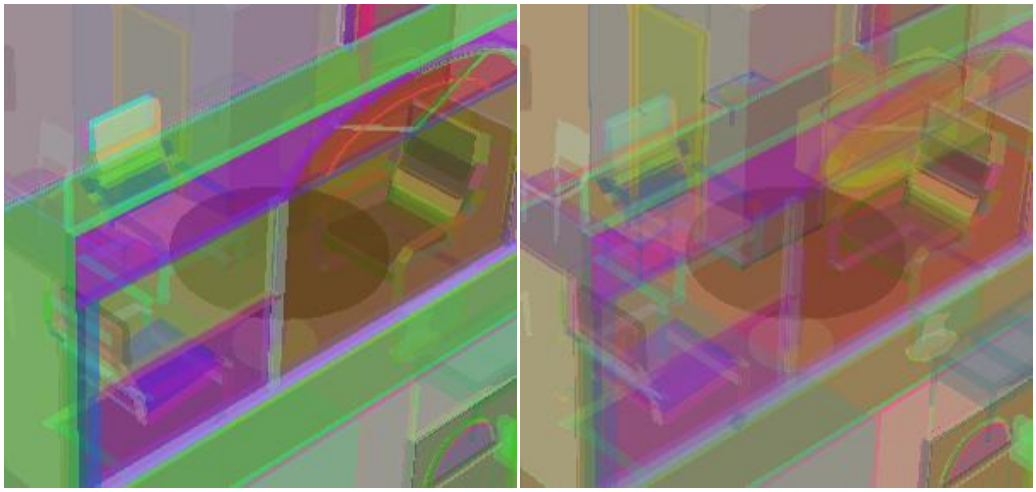


**Figura 4-27. Opacitat uniforme i opacitat decreixent per capa (per píxel).**

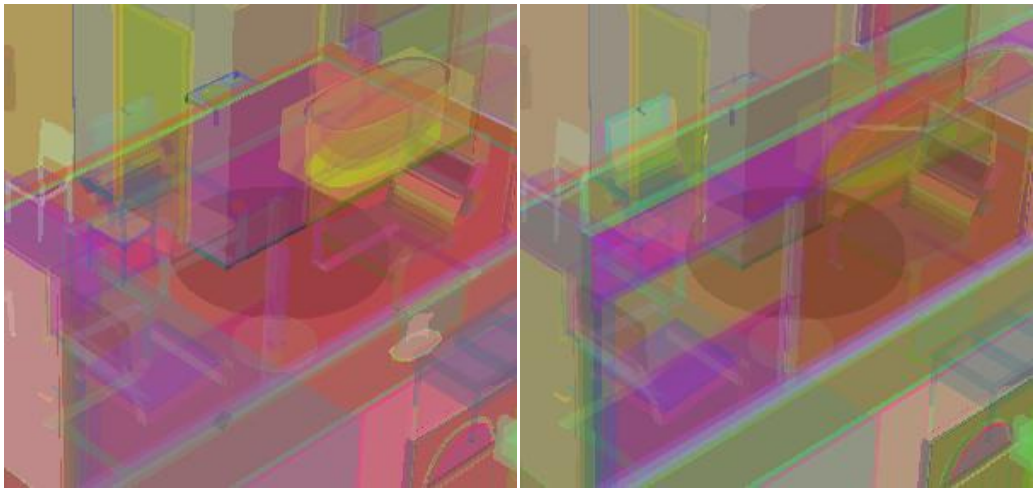


**Figura 4-28. Opacitat creixent per capa (per píxel) i opacitat decreixent per capa (per objecte).**

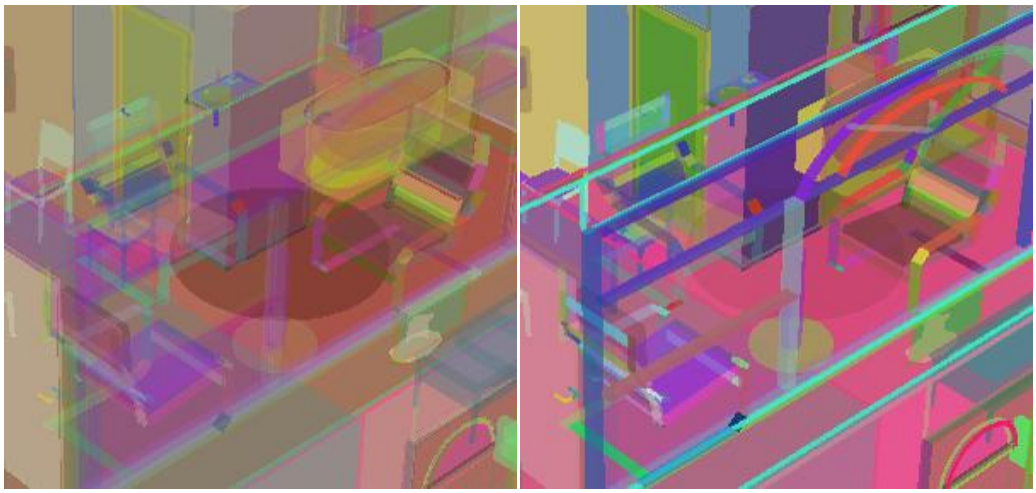


**Figura 4-29. Opacitat creixent per capa (per objecte) i optimització d'opacitat per objecte.**

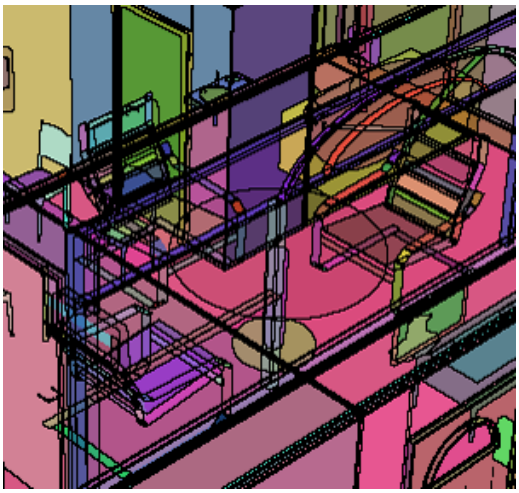


**Figura 4-30. Optimització d'opacitat per objecte amb contorn.**

## 4.3 RESULTATS DE L'HEURÍSTIC

S'han escollit les imatges mostrades anteriorment per ser avaluades per l'heurístic implementat. Per a cada model s'ha calculat: el nombre de píxels de les imatges, el nombre de píxels ben definits que hi ha a la imatge i el percentatge de píxels ben definits respecte als totals. Els resultats són els següents:

**Model d'una planta** Píxels totals 820633

| Tècnica | Píxels ben definits | Percentatge | Percentatge relatiu |
|---|---|---|---|
| Opacitat uniforme | 27555 | 3.36% | 24.17% |
| Opacitat decreixent per capa (per píxel) | 31197 | 3.80% | 27.34% |
| Opacitat creixent per capa (per píxel) | 35911 | 4.37% | 31.44% |
| Opacitat decreixent per capa (per objecte) | 29580 | 3.60% | 25.90% |
| Opacitat creixent per capa (per objecte) | 38337 | 4.67% | 33.60% |
| Optimització d'opacitat per objecte | 67426 | 8.21% | 59.06% |
| Optimització d'opacitat per objecte amb contorn | 114565 | 13.90% | 100% |

Taula 4-1

**Model de la casa** Píxels totals 820633

| Tècnica | Píxels ben definits | Percentatge | Percentatge relatiu |
|---|---|---|---|
| Opacitat uniforme | 35376 | 4.31% | 24.66% |
| Opacitat decreixent per capa (per píxel) | 36263 | 4.41% | 25.23% |
| Opacitat creixent per capa (per píxel) | 41786 | 5.09% | 29.12% |
| Opacitat decreixent per capa (per objecte) | 35336 | 4.30% | 24.60% |
| Opacitat creixent per capa (per objecte). | 45106 | 5.50% | 31.46% |
| Optimització d'opacitat per objecte | 85367 | 10.40% | 59.50% |
| Optimització d'opacitat per objecte amb contorn | 143415 | 17.48% | 100% |

Taula 4-2

Els resultats són bastant clars ja que les diferències de xifres són força elevades. En primer lloc, es pot comprovar que la tècnica d'opacitat uniforme genera els resultats més pobres, tot i que en el model de la casa no és el pitjor mentre que les tècniques que generen una opacitat creixent o decreixent per capa (tant per objecte com per píxel) tenen tots uns resultats força similars. Tot i això, com era d'esperar, les tècniques que assignen una opacitat més gran als objectes com més lluny estan donen uns resultats una mica superiors a la resta.

Tot i això, la tècnica d'optimització d'opacitat per objecte pràcticament dobla els resultats de les altres tècniques donant a pensar que el major nombre d'objectes identificables es troben mitjançant aquest algorisme. La millora del propi algorisme, al afegir-li els contorns també resulta força evident oferint un increment dels píxels ben definits de manera substancial.

## 4.4 TESTS D'USUARI

Per a verificar que els resultats obtinguts en l'apartat anterior són correctes s'ha optat per dur a terme unes petites proves amb usuaris. Les proves s'han distribuït en 2 blocs clarament diferenciats: en el primer d'ells els usuaris han d'identificar objectes senzills (taules, cadires, plantes) i comptar quantes instancies d'aquests es troben en cadascuna de les imatges. En el segon bloc les preguntes consisteixen en obtenir informació a partir de la localització espacial dels objectes. Preguntes del tipus "Quantes sales s'han de creuar per anar d'A a B?" o "Quantes instancies del objecte X hi ha a la sala que te 2 objectes Y?" entren dins d'aquest bloc.

Per tal de que el test fos ideal, caldria haver fet exactament les mateixes preguntes per a tots els models i tècniques a tots els usuaris. D'aquesta forma, però, el test es faria massa llarg i molest per als usuaris que al final podrien deduir les respostes enlloc de comptar correctament, problema conegut com a *learning*. Per a evitar-lo, hi ha 2 opcions, fragmentar la gran llista de preguntes i que cada part de preguntes les contesti un grup de persones diferents essent necessari un nombre de gent molt elevat o

realitzar petites modificacions al model per a que els usuaris no puguin intuir per les preguntes anteriors quants objectes hi ha a l'actual imatge.

La opció que s'ha triat és la segona en no disposar de la suficient gent necessària per poder triar la primera opció. S'ha modificat el model de les oficines modificant les posicions d'un despatx i els lavabos, quedant com a resultat 3 models d'oficines i un de la casa. Les modificacions han estan dissenyades per tal de preservar el nivell d'oclusió i la dificultat de les tasques a dur a terme pels usuaris.

Per poder realitzar les preguntes s'ha utilitzat la funcionalitat que ofereix *Google Docs* de generar formularis, permetent generar una pàgina web amb les preguntes i guardant els resultats al finalitzar el qüestionari en un full de càlcul de Google Docs on es poden estudiar les dades obtingudes, realitzar gràfiques, etc.

En total ha participat un conjunt de 21 persones, 13 de les quals són homes i 8 dones. La majoria dels usuaris tenen estudis universitaris i cap d'ells té problemes de visió relacionats amb la identificació de colors. En les figures 4.31 i 4.32 es poden veure les dades dels subjectes.

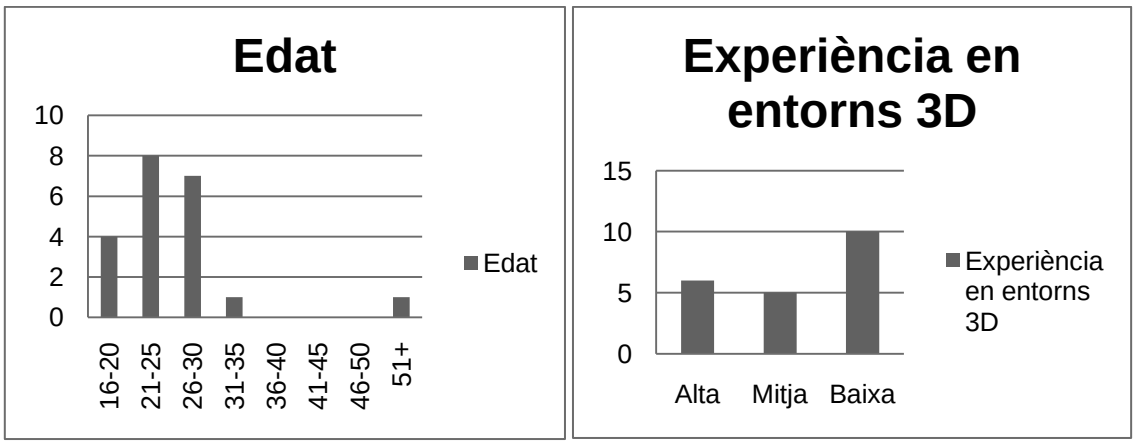



**Figura 4-31**

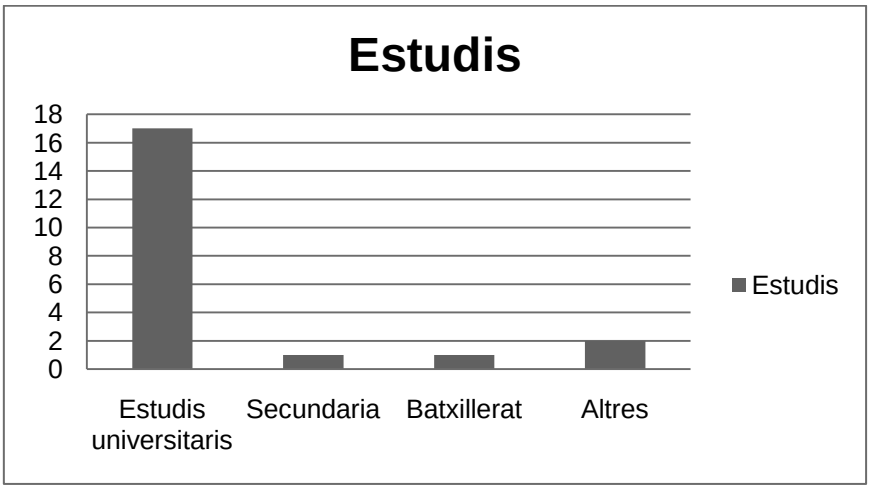



**Figura 4-32**

A continuació es mostren els resultats obtinguts agrupats per a cada una de les tècniques utilitzades. Les preguntes que estan enfocades a la identificació d'objectes són les preguntes des de la A fins la N, mentre que les preguntes que intenten obtenir informació de la posició espacial dels objectes són les preguntes O fins la T. Les preguntes que s'han realitzat es troben en l'annex 1.

| Tècnica | Resposta esperada | Resposta mitja obtinguda | Desviació estàndard |
|---|---|---|---|
| Opacitat uniforme | | | |
| Pregunta A | 7 | 4.19 | 1.36 |
| Pregunta K | 5 | 2.14 | 1.10 |
| Pregunta P | 4 | 3.57 | 1.43 |
| Opacitat decreixent per capa (per píxel) | | | |
| Pregunta B | 6 | 3.80 | 1.47 |
| Pregunta M | 6 | 5.43 | 1.16 |
| Pregunta O | 5 | 3.33 | 1.59 |
| Opacitat creixent per capa (per píxel) | | | |
| Pregunta D | 6 | 5.52 | 1.20 |
| Pregunta J | 5 | 4.62 | 0.92 |
| Opacitat decreixent per capa (per objecte) | | | |
| Pregunta E | 7 | 4.86 | 1.88 |

| | | | |
|---|---|---|---|
| Pregunta N | 5 | 3.81 | 1.50 |
| Pregunta Q | 2 | 1.05 | 0.74 |
| Pregunta R | 7 | 4.71 | 2.08 |
| Opacitat creixent per capa (per objecte) | | | |
| Pregunta F | 6 | 5.90 | 1.14 |
| Pregunta L | 7 | 6.57 | 1.47 |
| Optimització d'opacitat per objecte | | | |
| Pregunta G | 12 | 7.23 | 1.60 |
| Pregunta H | 6 | 5.24 | 0.77 |
| Pregunta T | 3 | 3.04 | 0.74 |
| Optimització d'opacitat per objecte amb contorn | | | |
| Pregunta C | 5 | 2.61 | 1.20 |
| Pregunta I | 7 | 7.24 | 1.14 |
| Pregunta S | 4 | 3.62 | 1.24 |

**Taula 4-3**

Els resultats obtinguts amb les preguntes als usuaris no són tan clars com els que s'han obtingut mitjançant l'heurístic. Per poder comparar els resultats s'han d'obviar, per ara, els resultats de les preguntes O a T, ja que el seu objectiu és l'avaluació de la posició espaial dels objectes.

Primer de tot cal destacar que les preguntes C, G, O i R donen uns resultats dolents. Això és degut a que són les preguntes en les quals sortia la casa de dues plantes, evidenciant d'aquesta manera, que la dificultat d'identificació d'objectes en aquests casos és molt més elevada que en les preguntes on només surt l'oficina.

Es pot veure que les diferències de resultats entre les tècniques són força petites però es pot veure que la tècnica amb pitjors resultats és la d'opacitat uniforme. Les tècniques que assignen opacitat decreixent per capa (per objecte i per píxel) tampoc tenen uns resultats del tot satisfactoris. Tal i com pronostica l'heurístic implementat, les tècniques que ofereixen uns resultats que mostren un major nombre d'objectes identificables als usuaris són les tècniques d'opacitat creixent per capa (per

píxel i per objecte), optimització d'opacitat per objecte i optimització d'opacitat per objecte amb contorn. Tot i això, les diferències entre els resultats de la tècnica d'optimització d'opacitat per objecte i les altres tècniques "bones" no són tan significatives com les que ofereix l'heurístic.

Respecte la identificació de la profunditat dels objectes, les tècniques que millor mostren la informació espaial són les d'opacitat uniforme, i la d'optimització d'opacitat per objecte, juntament amb la seva millora.

Ignorant el conjunt de preguntes que tenien un nivell de dificultat més elevat (les que tenien com a imatge la casa de dues plantes), s'han agrupat les respostes per tècnica i s'ha calculat la mitjana de la diferència de les respostes obtingudes amb les respostes esperades. El resultat d'aquest càlcul es troba en la taula 4-4. La taula indica clarament com la tècnica basada en optimitzar l'opacitat redueix l'error mitjà a les respostes, amb una desviació molt petita. També cal destacar que visualitzar els contorns no necessàriament millora el reconeixement des objectes; de fet, l'error va ser lleugerament superior respecte l'optimització de les opacitats sense contorns.

| **Tècnica** | **Mitja error** | **Desviació estàndard** |
|---|---|---|
| Opacitat uniforme | 2.29 | 1.36 |
| Opacitat decreixent en profunditat (per píxel) | 1.67 | 1.22 |
| Opacitat creixent en profunditat (per píxel) | 0.76 | 0.93 |
| Opacitat decreixent en profunditat (per objecte) | 1.65 | 1.27 |
| Opacitat creixent en profunditat (per objecte) | 0.83 | 1.03 |
| Optimització d'opacitat per objecte | **0.59** | **0.70** |
| Optimització d'opacitat per objecte amb contorn | 0.74 | 0.96 |

**Taula 4-4**

# 5 PLANIFICACIÓ I COST ECONÒMIC

El projecte ha estat realitzat durant el període d'un quadrimestre en el que s'ha realitzat una dedicació completa (5 dies a la setmana i 8 hores al dia). El projecte ha passat per diverses fases que es mostren en la figura 5-1 juntament amb la seva duració i que s'expliquen en detall a continuació:

- Recerca d'informació: En aquesta primera fase, s'ha buscat tota la informació possible dels treballs que s'han realitzat sobre el tema que tracta aquest projecte i s'han estudiat els diferents mètodes trobats.
- Disseny i implementació de l'esquelet de l'aplicació: Per a poder implementar les tècniques, s'ha hagut de desenvolupar un esquelet d'una aplicació per poder, més tard, implementar les tècniques.
- Disseny i implementació de l'algorisme de *depth peeling*: En aquesta fase, s'ha dissenyat i implementat l'algorisme de *depth peeling* descrit en l'apartat 3.2.
- Disseny i implementació de les tècniques: La fase més llarga i més costosa ha sigut el disseny i implementació de les diferents tècniques utilitzades.
- Disseny i implementació de l'heurístic: En la última fase d'implementació s'ha dissenyat i desenvolupat l'heurístic descrit a l'apartat 3. 4.
- Realització de proves amb els usuaris: Durant aquesta fase, s'han preparat els models, les preguntes i els qüestionaris que realitzarien els usuaris. S'han realitzat les proves i s'han recollit els resultats obtinguts.
- Redacció de la memòria: La redacció de la memòria s'ha dut a terme en la última fase del projecte, un cop finalitzat el gruix de feina.

Per a dur a terme un control de la feina realitzada, s'han realitzat reunions setmanals amb el tutor del projecte, en les quals s'han mostrat els objectius complerts la setmana anterior, s'han establert els objectius a complir en la següent setmana i finalment s'han ajustat els objectius a llarg termini per tal de complir el calendari.

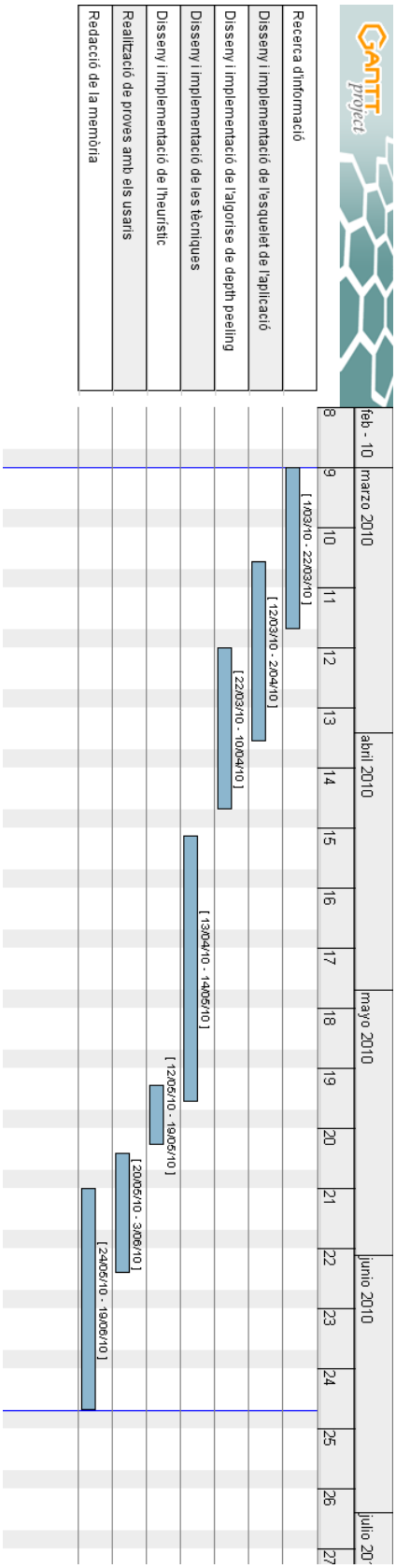

GANTT project
Recerca d'informació
Disseny i implementació de l'esquelet de l'aplicació
Disseny i implementació de l'algorisme de depth peeling
Disseny i implementació de les tècniques
Disseny i implementació de l'heurístic
Realització de proves amb els usaris
Redacció de la memòria
feb - 10
marzo 2010
abril 2010
mayo 2010
junio 2010
julio 20
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
[ 1/03/10 - 22/03/10 ]
[ 12/03/10 - 2/04/10 ]
[ 22/03/10 - 10/04/10 ]
[ 13/04/10 - 14/05/10 ]
[ 12/05/10 - 19/05/10 ]
[ 20/05/10 - 3/06/10 ]
[ 24/05/10 - 19/06/10 ]

**Figura 5-1**

Per calcular el cost econòmic del projecte s'han desglossat els diferents apartats del projecte, assignat els rols que corresponen a cada apartat i el nombre d'hores que s'ha dedicat a cada fase. Finalment, segons el rol i el temps estimat dedicat, s'ha assignat un preu. En la taula 5-1 es pot veure el resultat d'aplicar aquests càlculs. En el cost del projecte no s'ha tingut el compte el temps dedicat a la recerca d'informació ni a la redacció de la memòria.

| **Fase** | **Rol** | **Hores estimades** | **Cost/ hora** | **Cost final** |
|---|---|---|---|---|
| Disseny i implementació de l'aplicació | | | | |
| Anàlisi, especificació i disseny | Analista | 40 | 45€ | 1800€ |
| Implementació i proves | Programador | 88 | 30€ | 2640€ |
| Disseny i implementació del *depth peeling* | | | | |
| Disseny | Analista | 40 | 45€ | 1800€ |
| Implementació i proves | Programador | 88 | 30€ | 2640€ |
| Disseny i implementació de les tècniques | | | | |
| Disseny | Analista | 80 | 45€ | 3600€ |
| Implementació i proves | Programador | 112 | 30€ | 3360€ |
| Disseny i implementació de l'heurístic | | | | |
| Disseny | Analista | 16 | 45€ | 720€ |
| Implementació i proves | Programador | 24 | 30€ | 720€ |
| Realització de les proves d'usuari | | | | |
| Realització del formulari | Analista | 24 | 45€ | 1080€ |
| Anàlisi dels resultats | Analista | 24 | 45€ | 1080€ |
| | | | **Cost final** | 19440€ |

**Taula 5-1**

# 6 CONCLUSIONS

Els objectius del projecte han estat assolits amb èxit. S'han proposat un conjunt de tècniques que treballen amb objectes semi transparents oferint uns resultats que milloren la tècnica bàsica d'opacitat uniforme. A més, s'ha implementat un heurístic que permet la comparació entre diferents imatges i comparant els resultats amb els obtinguts amb les proves amb els usuaris es pot establir que l'heurístic té un funcionament força bo.

A part de l'estudi de les tècniques, en la creació de l'aplicació que gestiona les tècniques, s'han implementat diverses funcionalitats explicades de forma superficial en diferents assignatures, fent-les funcionar totes conjuntament i de forma correcta, obtenint una molt bona experiència en aquest aspecte.

Finalment, en l'aspecte personal, he quedat molt satisfet amb la feina feta. Un projecte d'aquest tipus, en el que es treballa en un tema sobre el que no s'ha investigat massa comporta certs riscs degut a la falta. A més, he pogut aplicar els coneixements que s'han anat adquirint durant la carrera tant en l'aspecte tècnic com en la planificació de la feina i la fixació d'objectius factibles.

## 6.1 EXTENSIONS

Hi ha diversos aspectes sobre els que es pot continuar treballant per millorar encara més els resultats obtinguts. El primer d'ells consisteix en l'aplicació d'iluminació a les escenes, que aporta una sensació de volum facilitant d'aquesta manera la identificació d'elements. A continuació es mostren 3 exemples on es veu la figura de Lego i en les altres 2 hi ha la imatge del torus,con,cub i esfera i les oficines, ambdues amb il·luminació per vèrtex i implementant la tècnica *d'alpha blending*.

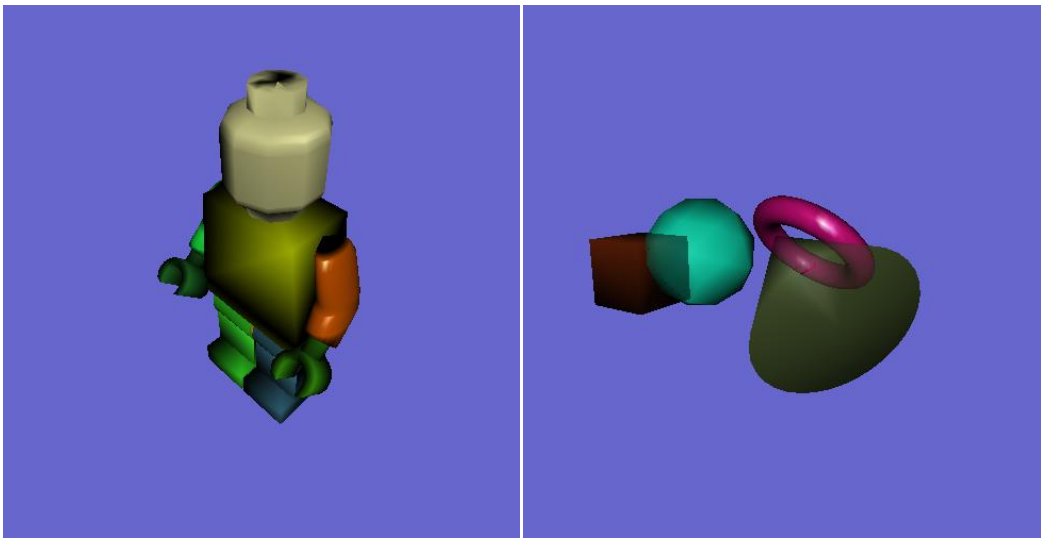


Figura 6-1

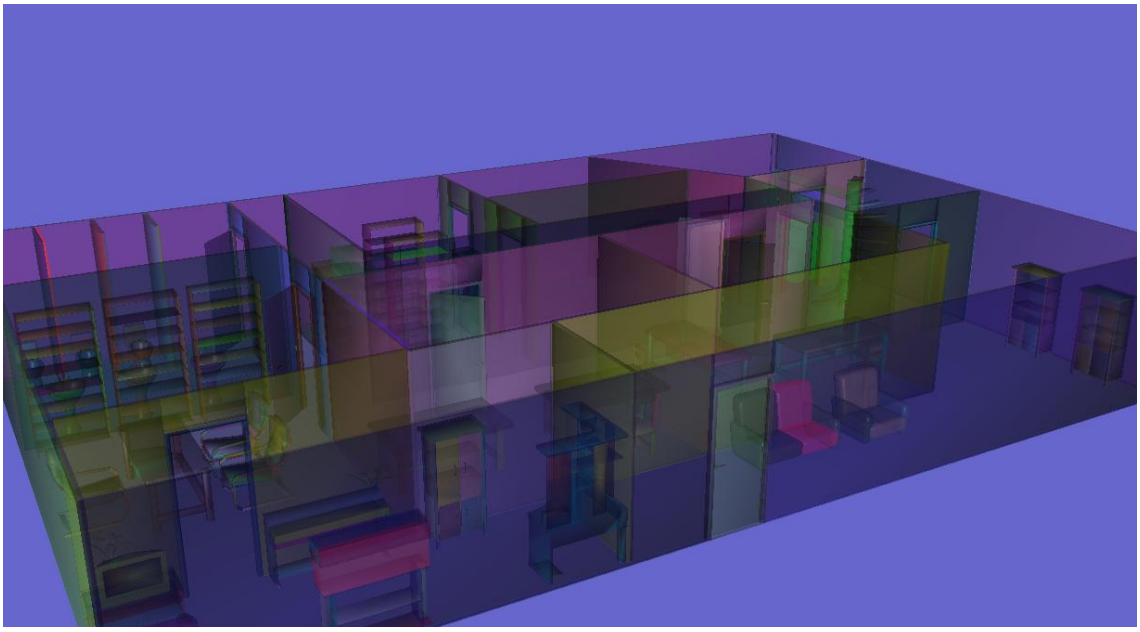


Figura 6-2

Un altre canvi que pot ser interessant d'investigar és l'heurístic. Amb altres estratègies per definir que és un píxel ben definit o bé fent les comprovacions mirant d'altres píxels veïns enlloc del que està a la dreta i a sobre del píxel que s'està tractant, poden variar els resultats obtinguts aconseguint, potser, un resultat millor.

# 7 BIBLIOGRAFIA

## Documents principals

C. Everitt. Interactive Order-independent transparency. *Technical report,* nVIDIA Corporation (2001).

N. Elmqvist. BalloonProbe: Reducing occlusion in 3D using interactive space distorsion. *Proceedings of de ACM Symposium on Virtual Reality Software and Technology 2005*, pp. 134-137, 2005.

N. Elmqvist, P. Tsigas. A taxonomy of 3D occlusion management for visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1095-1109, 2008.

N. Elmqvist, U. Assarsson, P. Tsigas. Employing dynamic transparency for 3D occlusion management: Design issues and evaluation. *Proceedings of INTERACT*, ser. LNCS, C. Baranauskas, P. Palanque, J. Abascal, S. D. J. Barbosa, Eds., vol. 4662, pp. 532-545, 2007.

U. Assarsson, N. Elmqvist, P. Tsigas. Image-space dynamic transparency for improved 3D object discovery. *Technical report* 2006:10, Chalmers University of Technology, 2006.

## Altres

Enllaços vàlids en la data de Juny de 2010.

Google Docs: Paquet d'ofimàtica de Google.

- http://docs.google.com

Google Scholar: Cercador d'articles científics i tècnics.

- http://scholar.google.es/

Lighthouse 3D: Tutorials d'OpenGL.

- http://www.lighthouse3d.com/opengl/

NeHe: Tutorials d'OpenGL.

- http://nehe.gamedev.net/

Songho: Tutorials d'OpenGL.

- http://www.songho.ca/opengl/index.html

QT4.3: Guía de referència.

- http://doc.trolltech.com/4.3/index.html

SweetHome 3D: Aplicació que genera models de cases/oficines.

- http://www.sweethome3d.com/es/index.jsp

Wikipedia: Enciclopèdia *on-line.*

- http://en.wikipedia.org/wiki/Main_Page

# 8 ANNEX 1: PREGUNTES DEL TEST D'USUARI

Per evitar confusions, abans de començar cada experiment es mostraven un conjunt d'imatges en les que es veuen els objectes a identificar en les preguntes del test. Els objectes que s'han d'identificar són els següents: cadires, taules i plantes.

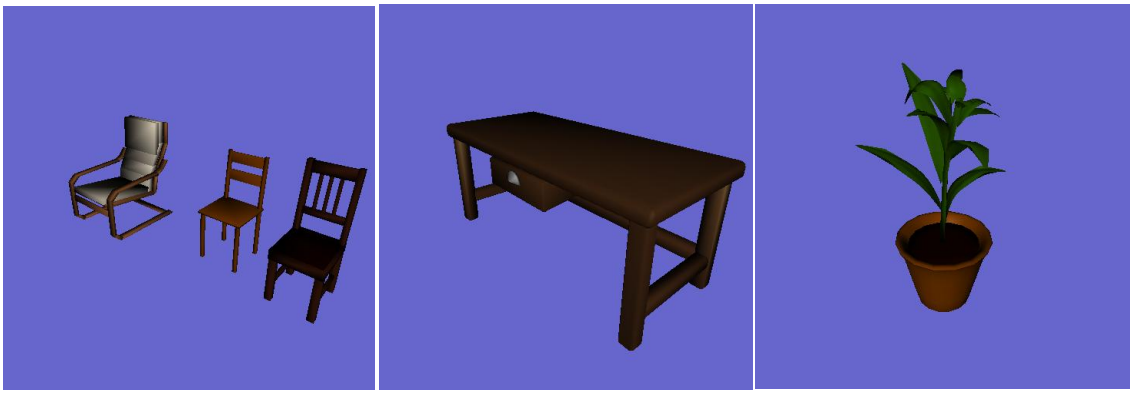


Figura 8-1

**Pregunta A:**

Comptar el nombre de cadires de la imatge.

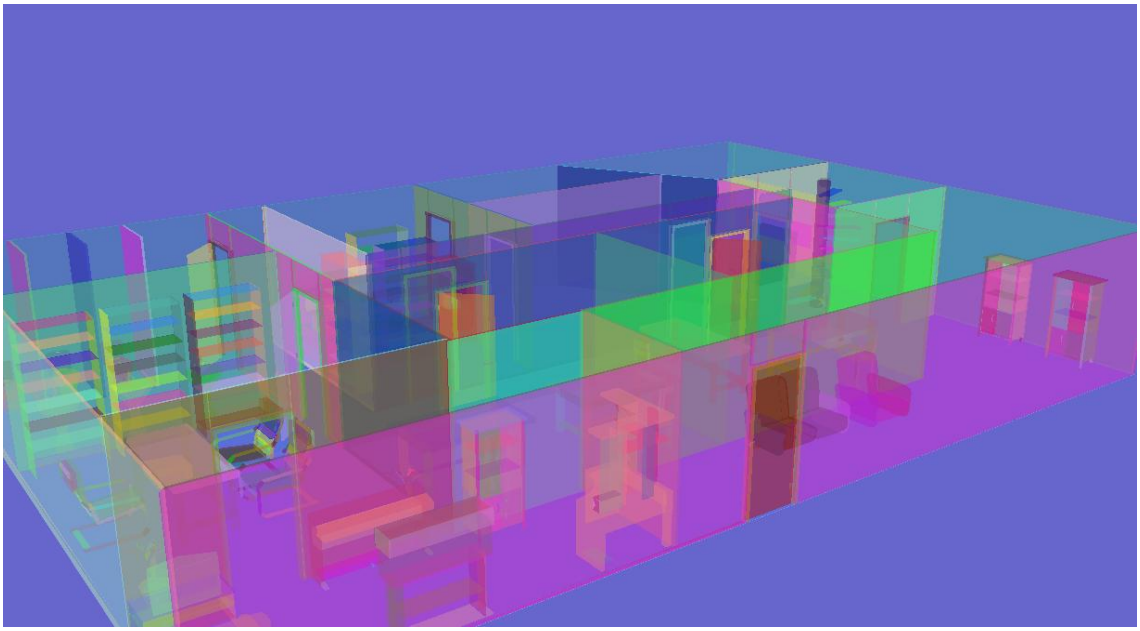


Figura 8-2

**Pregunta B:**

Comptar el nombre de plantes de la imatge.

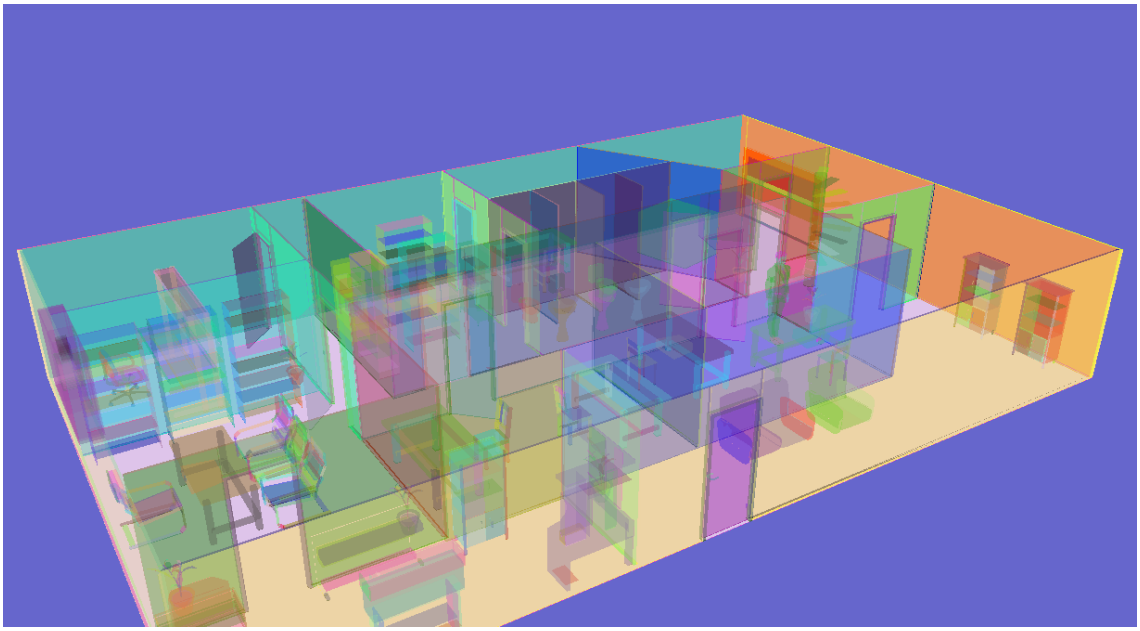


Figura 8-3

**Pregunta C:**

Comptar el nombre de taules de la imatge. En aquest cas, s'ha de comptar qualsevol tipus de taula.

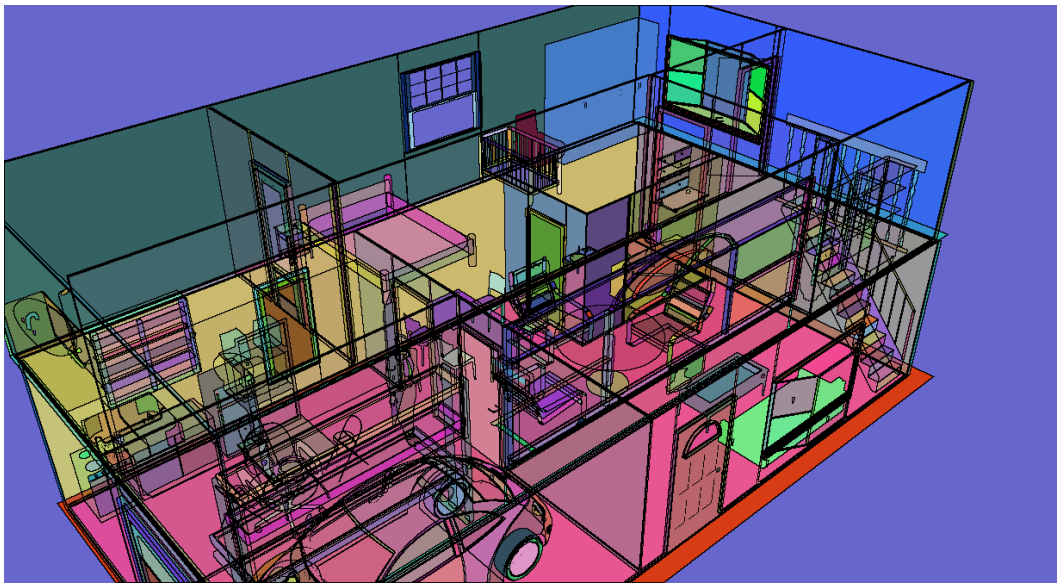


Figura 8-4

**Pregunta D:**

Comptar el nombre de taules de la imatge.

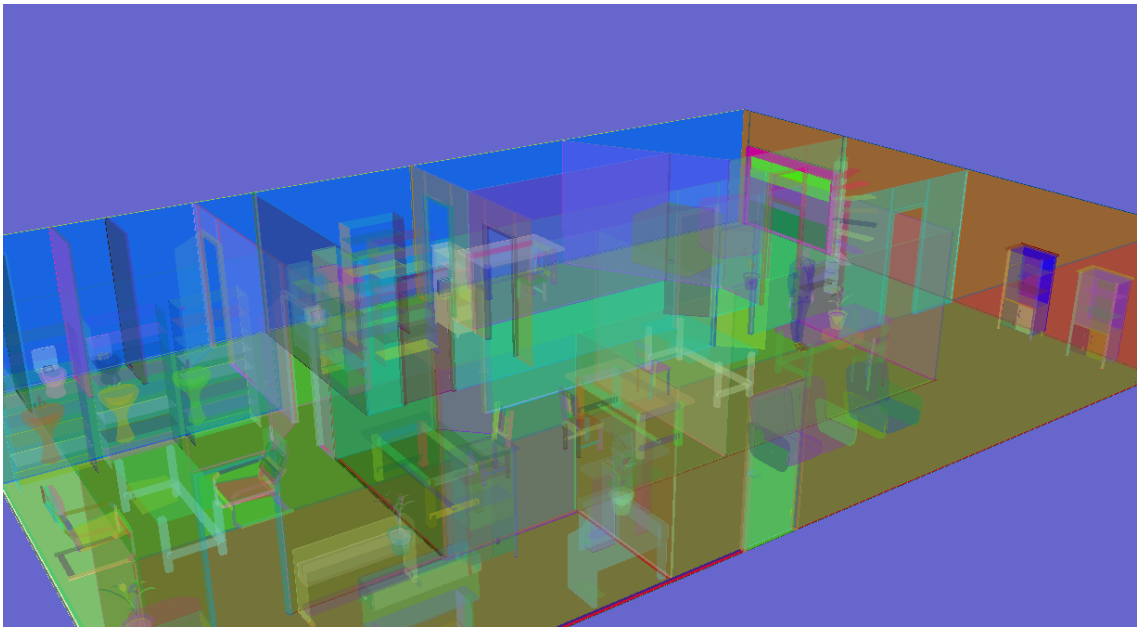


Figura 8-5

**Pregunta E:**

Comptar el nombre de cadires que hi ha a la imatge.

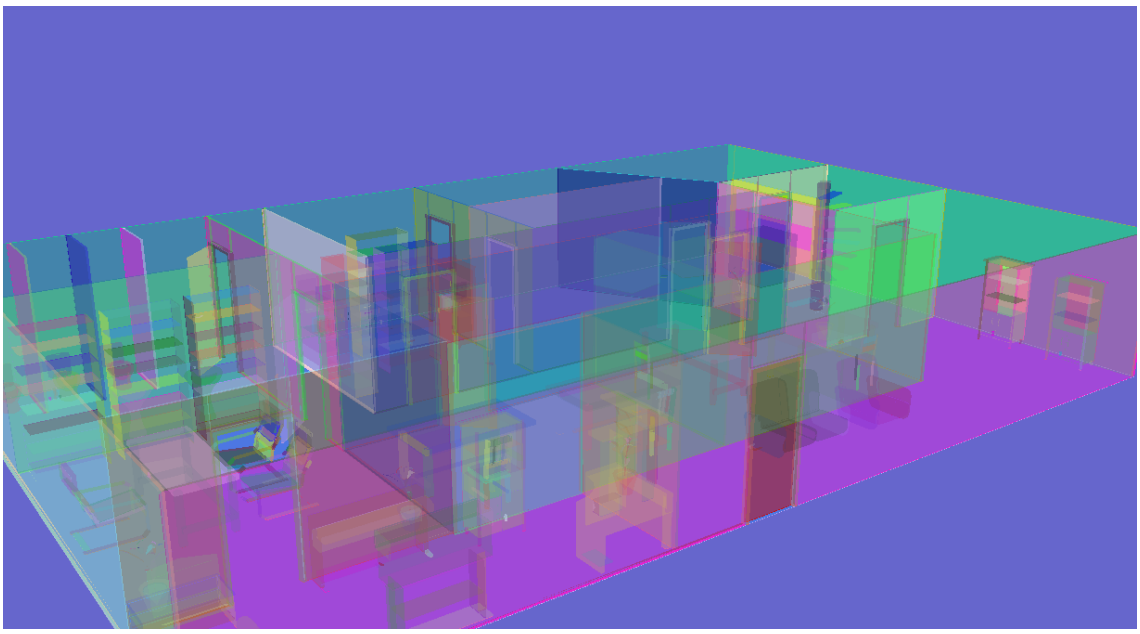


Figura 8-6

**Pregunta F:**

Comptar el nombre de taules de la imatge.

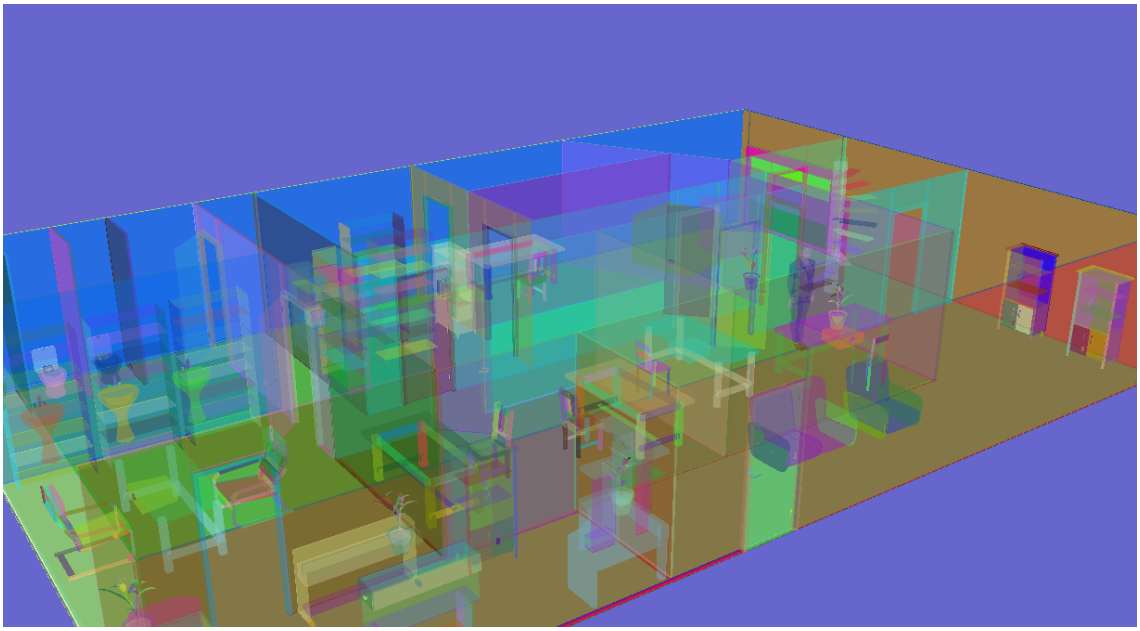


Figura 8-7

**Pregunta G:**

Comptar el nombre de cadires que hi ha a la imatge. En aquest cas, s'ha de comptar qualsevol tipus de cadira, tamboret o sofà.

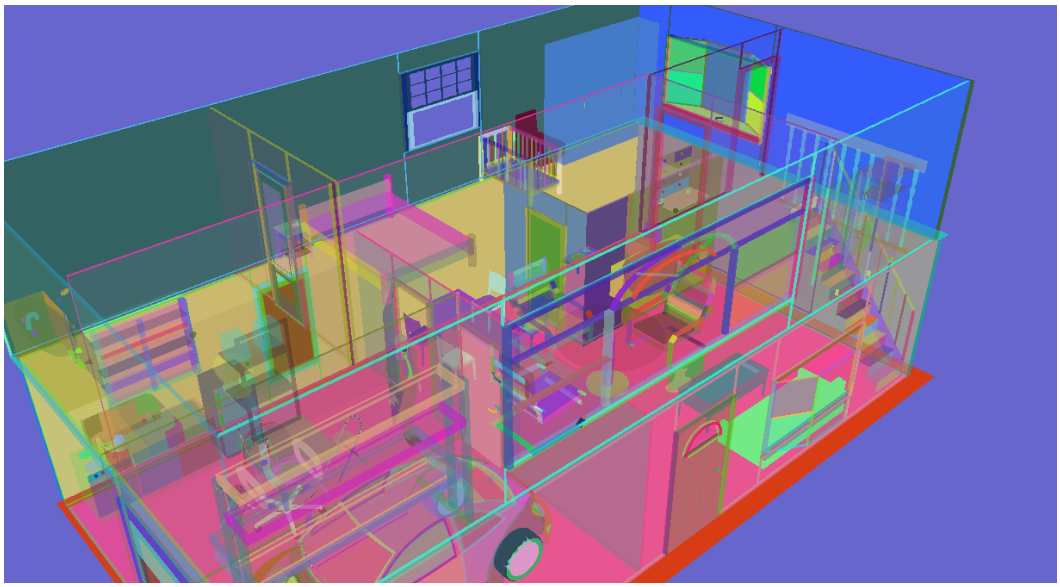


Figura 8-8

**Pregunta H:**

Comptar el nombre de plantes de la imatge.

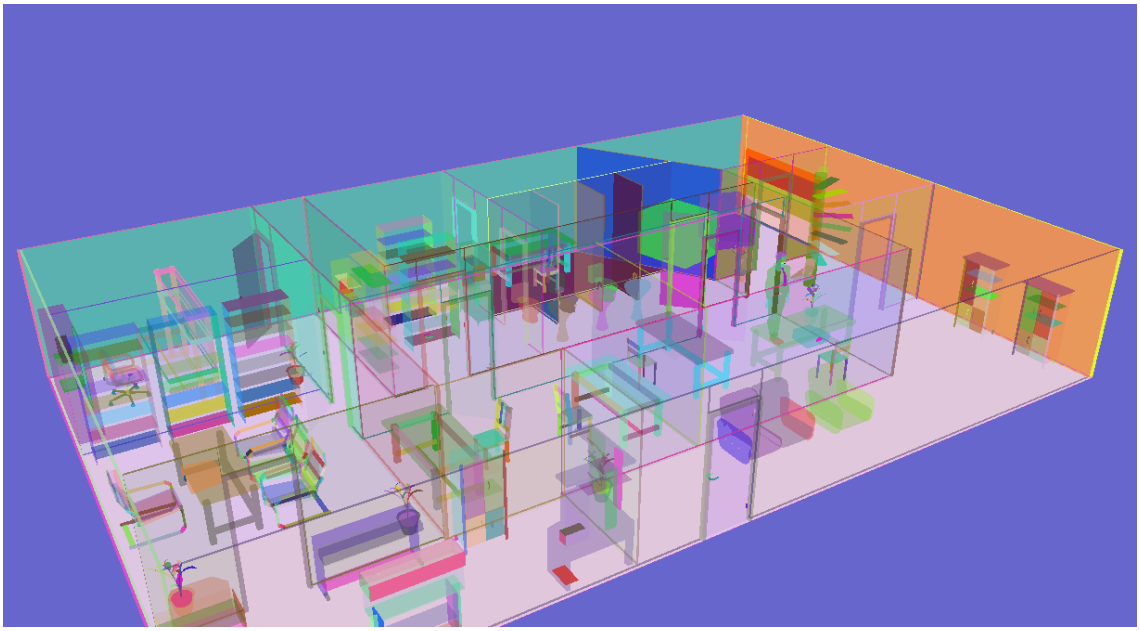


Figura 8-9

**Pregunta I:**

Comptar el nombre de cadires que hi ha a la imatge.

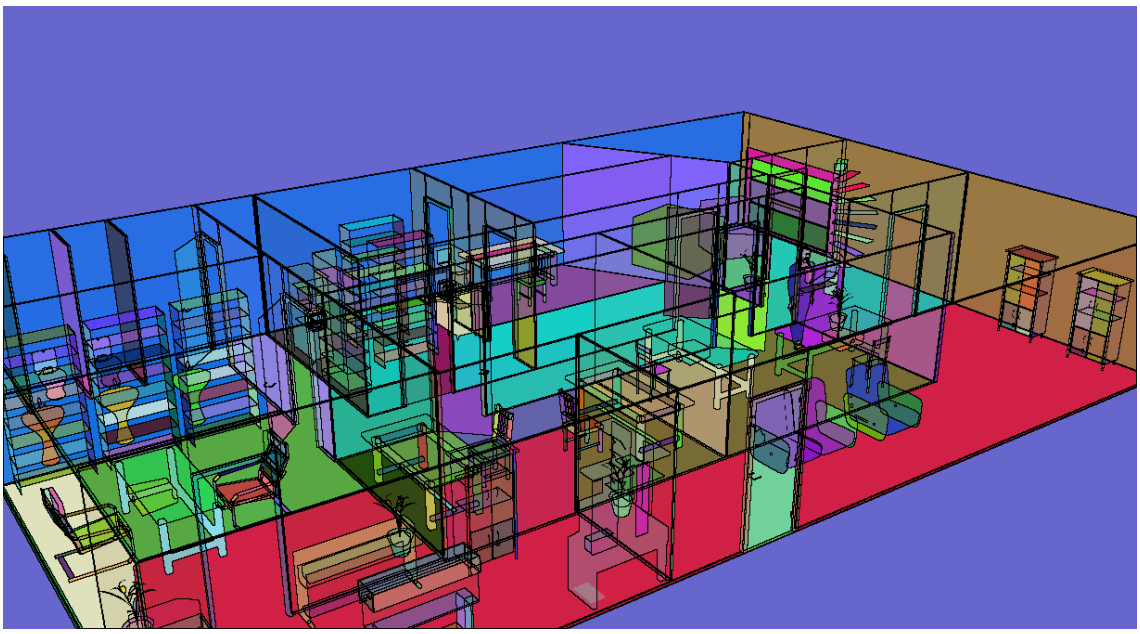


Figura 8-10

**Pregunta J:**

Comptar el nombre de taules de la imatge.

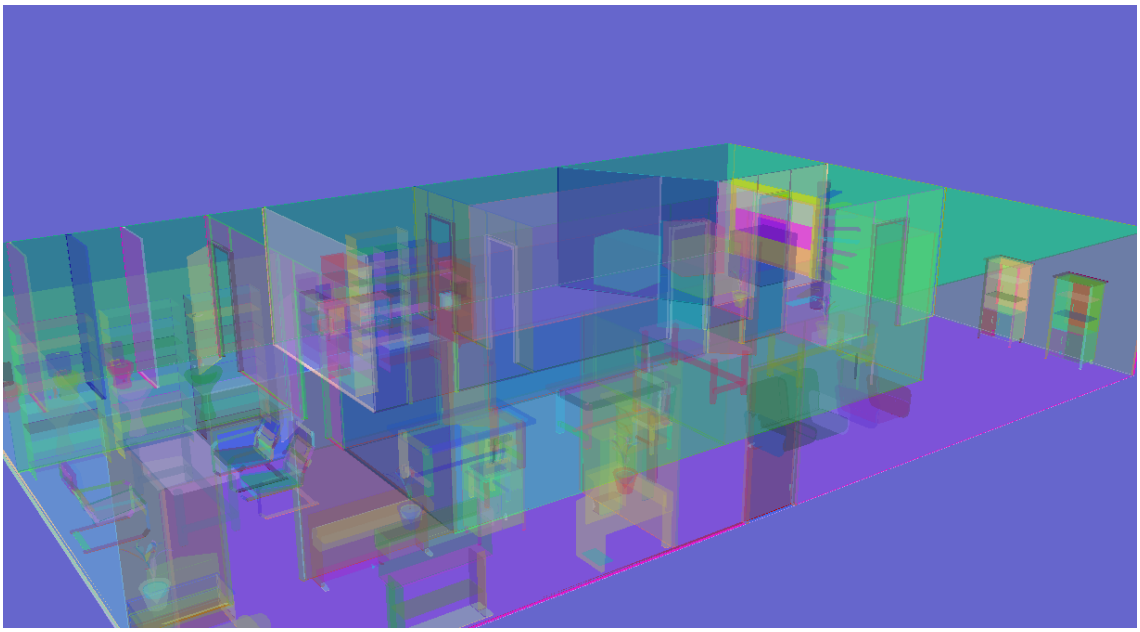


Figura 8-11

**Pregunta K:**

Comptar el nombre de plantes de la imatge.

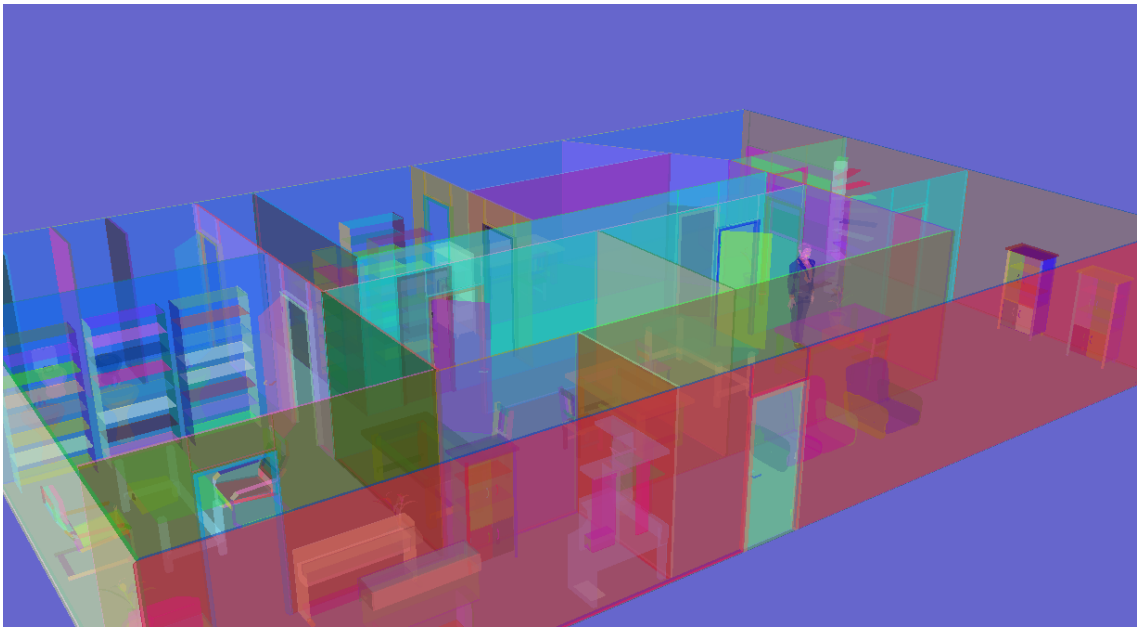


Figura 8-12

**Pregunta L:**

Comptar el nombre de cadires que hi ha a la imatge.

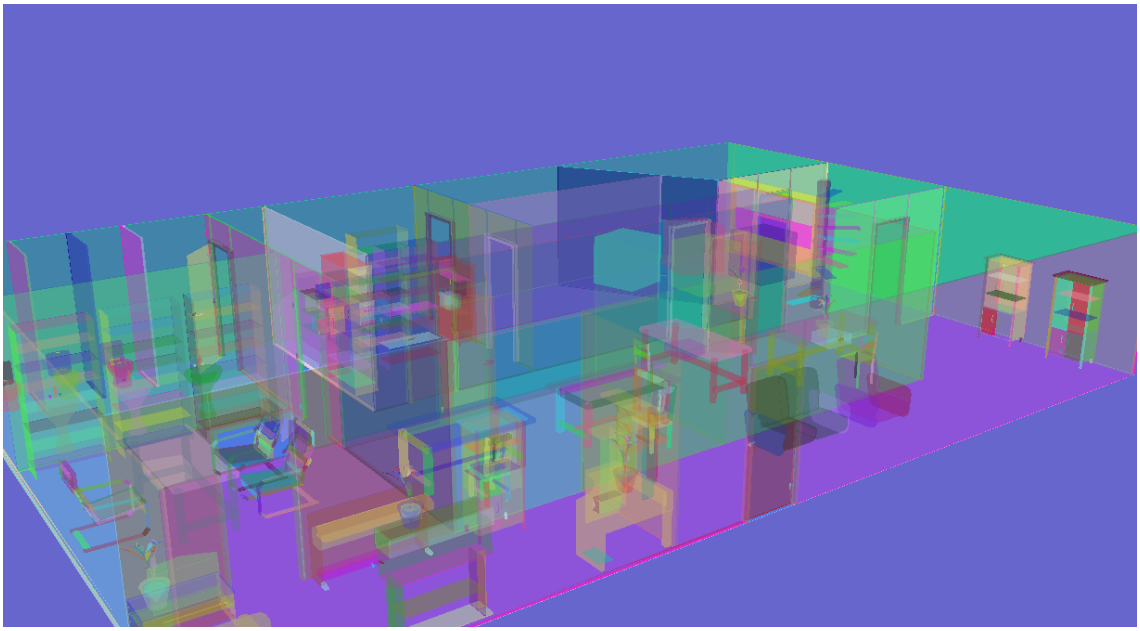


Figura 8-13

**Pregunta M:**

Comptar el nombre de taules de la imatge.

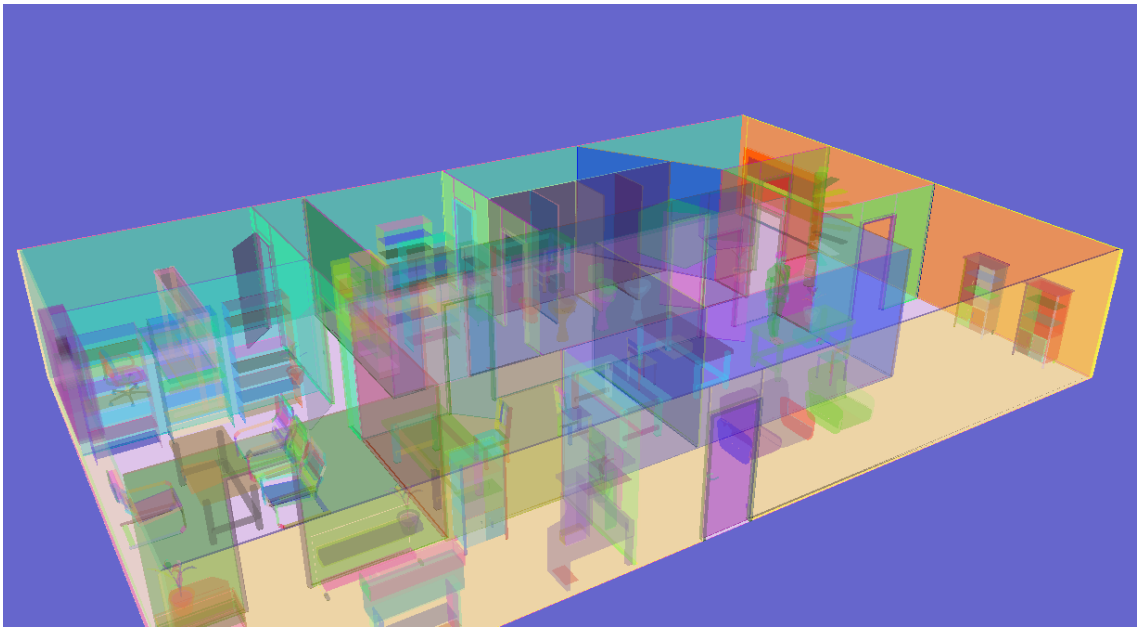


Figura 8-14

**Pregunta N:**

Comptar el nombre de taules de la imatge.

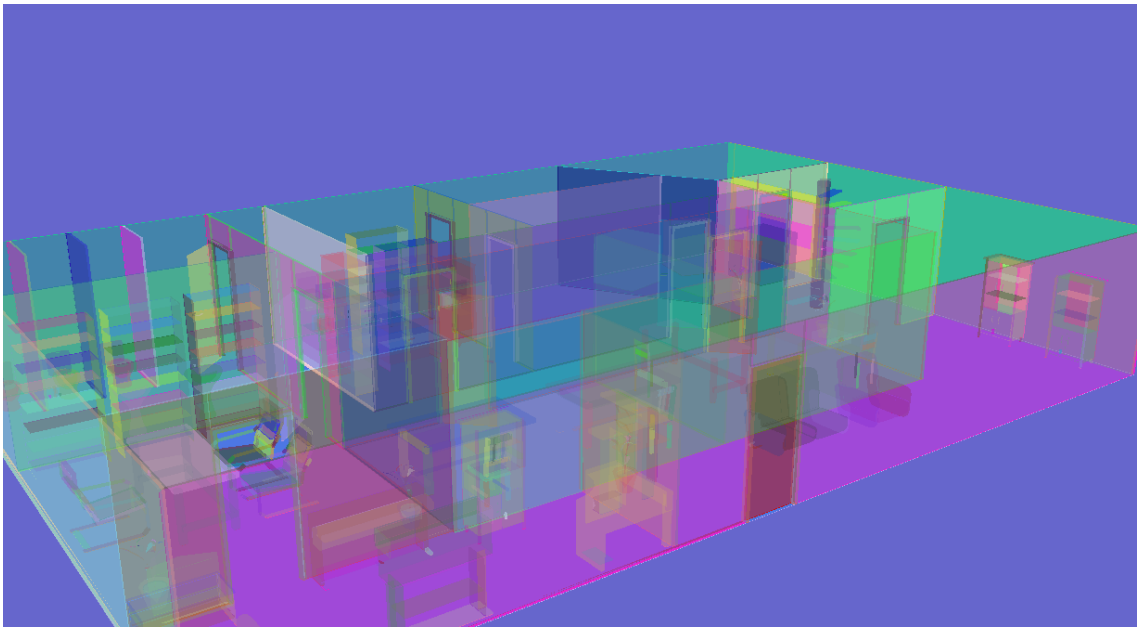


Figura 8-15

**Pregunta O:**

Comptar el nombre de cadires que es troben en la planta superior. En aquest cas, s'ha de comptar qualsevol tipus de cadira, tamboret o sofà.

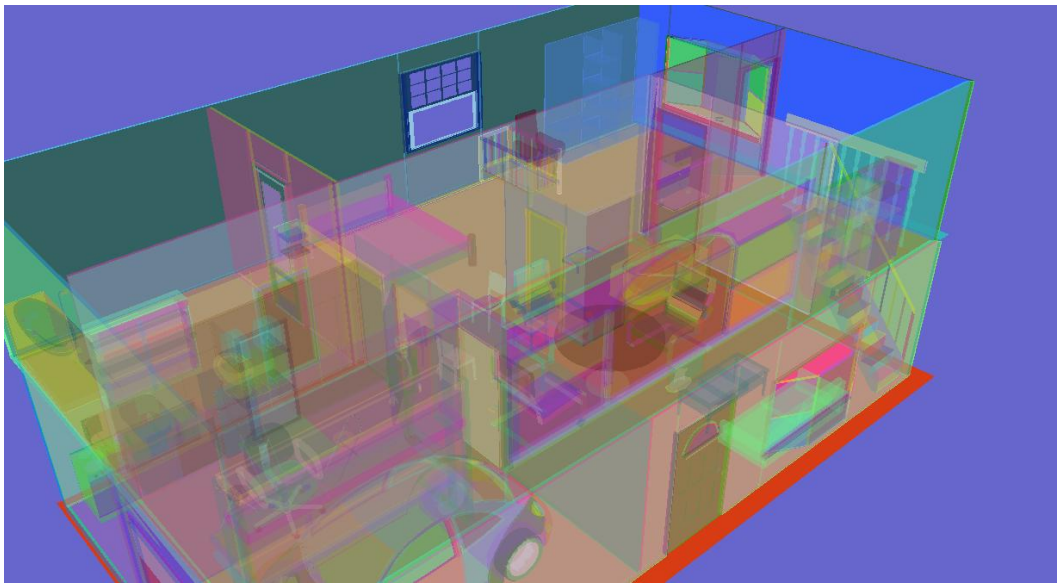


Figura 8-16

**Pregunta P:**

Comptar el nombre d'elements de la sala on es troba l'home. Els elements a comptar són: taules, cadires i plantes.

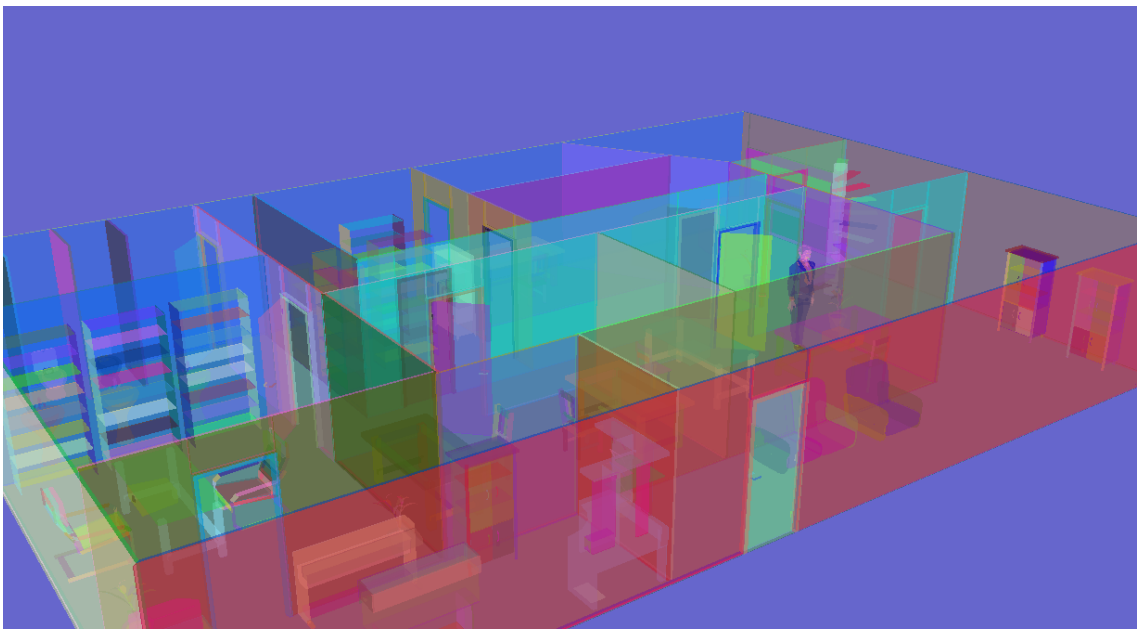


Figura 8-17

**Pregunta Q:**

Comptar el nombre de cadires de la sala on es troba l'home.

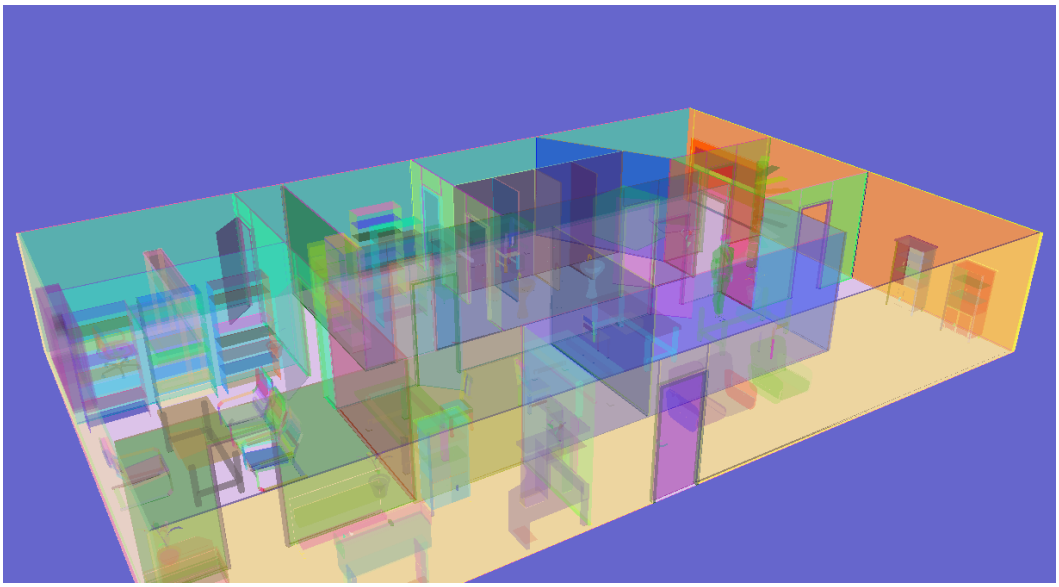


Figura 8-18

**Pregunta R:**

Comptar el nombre de cadires que es troben en la planta inferior. En aquest cas, s'ha de comptar qualsevol tipus de cadira, tamboret o sofà.

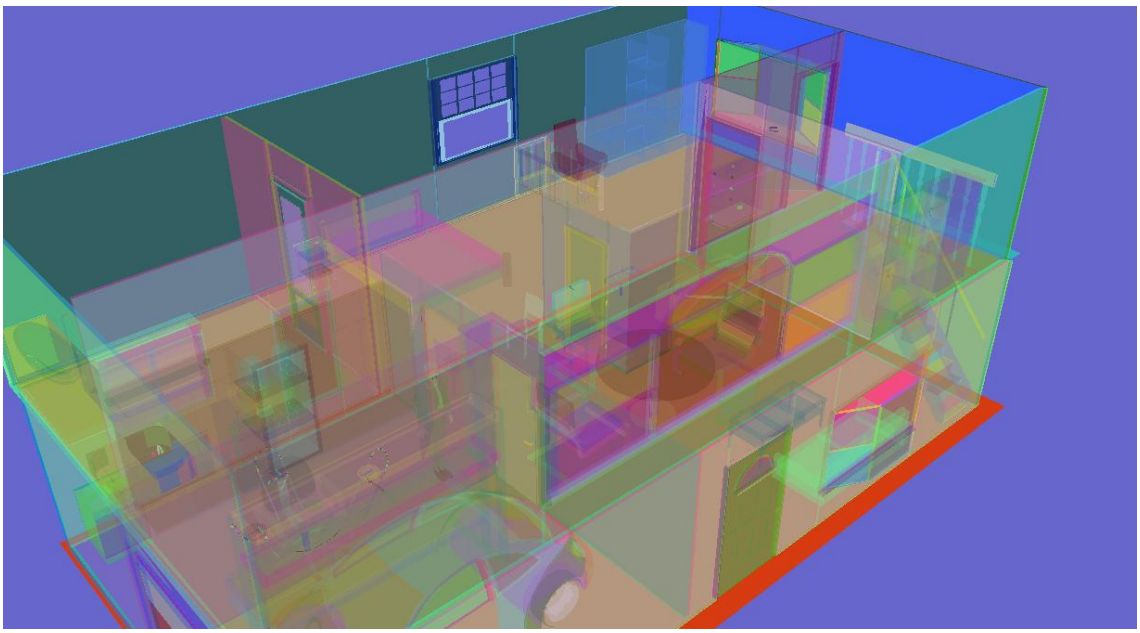


**Figura 8-19**

**Pregunta S:**

Comptar el nombre de sales que ha de creuar l'home per sortir de les oficines. S'ha de tenir en compte també que s'han de comptar la sala on està l'home, la sala en la que es troba la porta de sortida i que el passadís és considerat una sala.

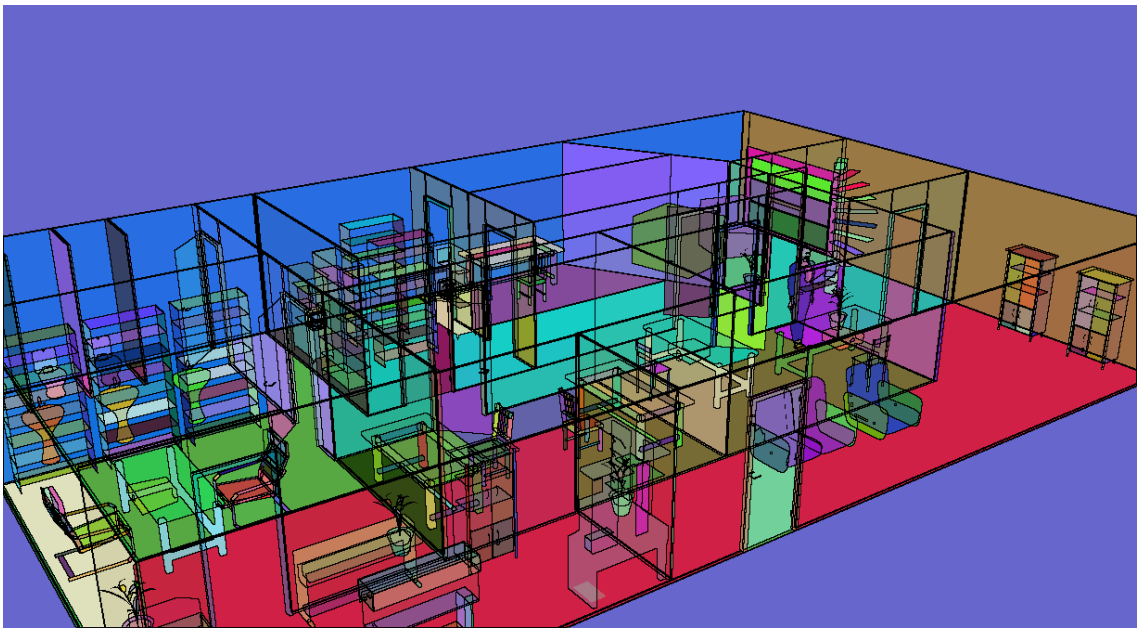


Figura 8-20

**Pregunta T:**

Comptar el nombre de sales que ha de creuar l'home per anar a la sala que només te una planta en el seu interior. S'ha de tenir en compte també que s'han de comptar la sala on està l'home, la sala destí i que el passadís és considerat una sala.

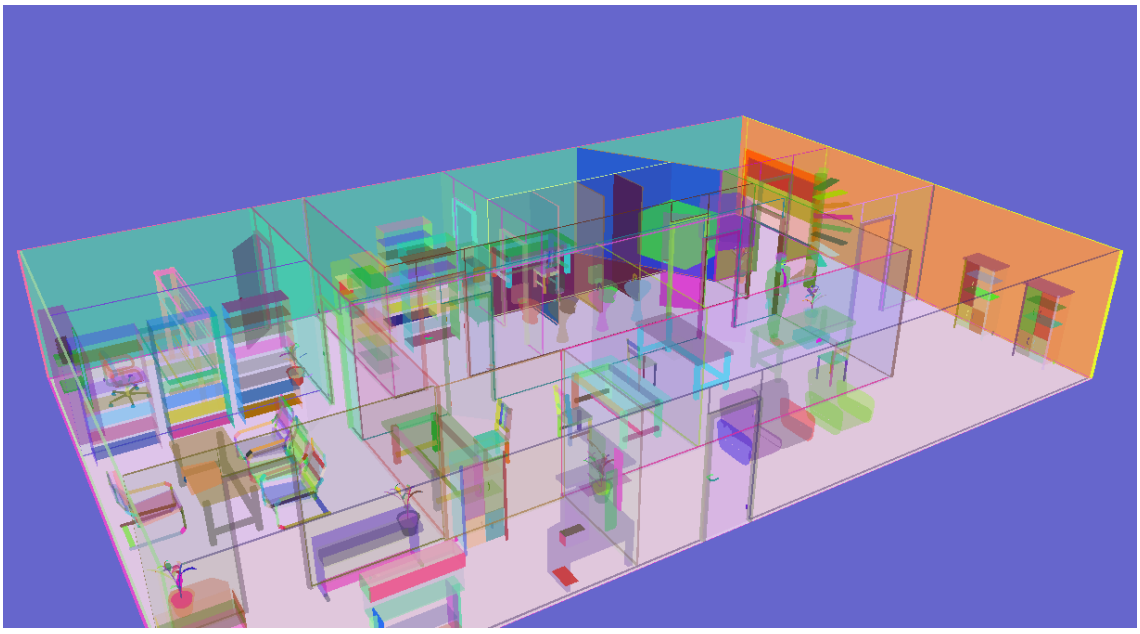


**Figura 8-21**

# 9 ANNEX 2: MANUAL D'USUARI DE L'APLICACIÓ

En aquest apartat s'explica com utilitzar l'aplicació desenvolupada per a generar les imatges resultants de les diferents tècniques. A l'executar l'aplicació es mostra una imatge com la que es veu a continuació (figura 9-1).

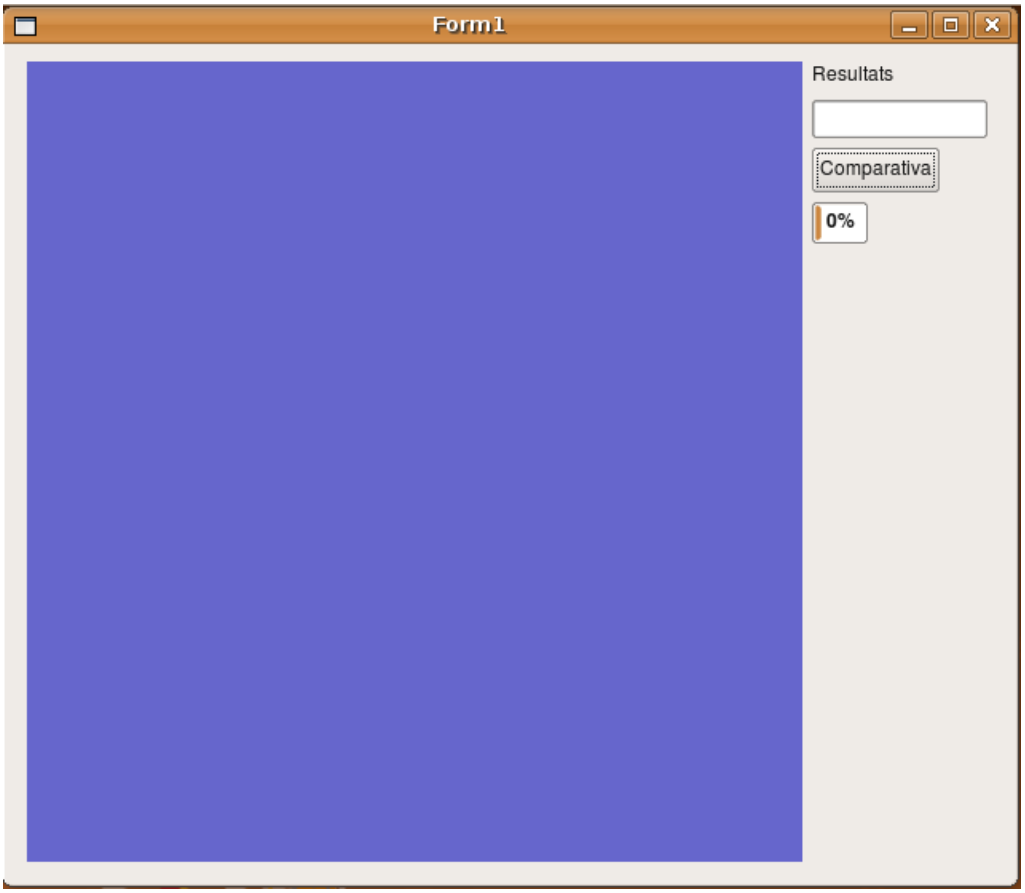



**Figura 9-1**

Com es pot veure, hi ha pocs botons, donat que la majoria de comandes s'introdueixen mitjançant el teclat. Primer de tot, per poder treballar s'ha d'obrir algun model existent pressionant la tecla **L**, moment en el qual apareix una finestra (figura 9-2) en la que es mostren tots els models existents per seleccionar el que es vulgui.

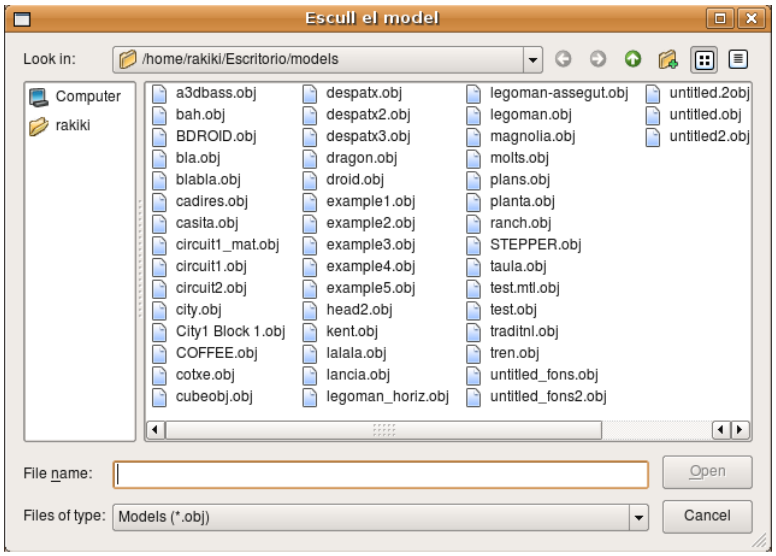



**Figura 9-2**

Un cop està el model carregat és pot interactuar amb el model. La guia de les tecles i quina és la seva funció es mostra a continuació:

- **F**: Mostra el model en filferros.
- **S**: Mostra el model en mode sòlid.
- **P**:Activa/Desactiva els colors aleatoris per objecte. Cal tenir els colors aleatoris activats per poder executar la obtenció de les imatges.
- **Botó esquerre del ratolí**: Amb el botó esquerre del ratolí polsat i movent-lo, es pot rotar el model.
- **Botó esquerre del ratolí + *shift***: Utilitzant aquesta combinació mentre es mou el ratolí es realitza un zoom.
- **Botó dret del ratolí + *shift***: Utilitzant aquesta combinació mentre es mou el ratolí es realitza un *pan*.
- **D**: Genera el conjunt d'imatges de les tècniques calculades mitjançant la tècnica de *depth peeling* i les guarda en la carpeta *images*. A continuació es mostren el nom de fitxer que te el resultat de cada tècnica:
    - Opacitat uniforme: **solucio1.bmp**
    - Opacitat decreixent per capa (per píxel): **solucio2.bmp**
    - Opacitat creixent per capa (per píxel): **solucio3.bmp**
    - Opacitat decreixent per capa (per objecte): **solucio4.bmp**
    - Opacitat creixent per capa (per objecte): **solucio5.bmp**

- **6**: Genera les solucions de la tècnica d'optimització d'opacitat per objecte. Com que el temps que es triga en generar aquestes imatges es força elevat, s'ha afegit una barra de progrés a la interfície gràfica per indicar a l'usuari quin percentatge s'ha completat. Al igual que en les imatges anteriors, s'emmagatzema el resultat en la carpeta *images* i els noms que tenen els fitxers són els següents:
    - Optimització d'opacitat per objecte: **solucio6.bmp**
    - Optimització d'opacitat per objecte amb contorn: **solucio7.bmp**
- **Botó Comparativa**: Un cop es tenen les 7 imatges, al polsar el botó que està etiquetat amb el nom de comparativa, executarà la comparativa entre les 7 imatges utilitzant l'heurístic, mostrant la llista ordenada de la imatge que te menys píxels ben definits a la imatge que te més píxels més ben definits. A més, en la terminal es mostra per cada imatge, el nombre de píxels totals, el nombre de píxels ben definits i el percentatge de píxels ben definits. A continuació es mostra una imatge amb un exemple:

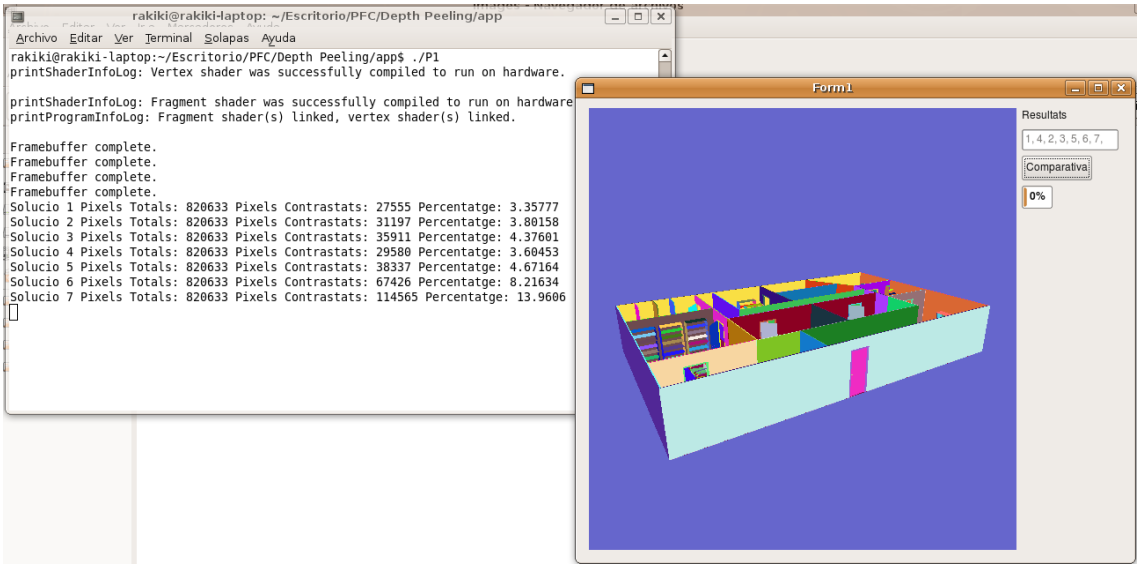



**Figura 9-3**