

Gràfics i Visualització de Dades

T2: Sistemes gràfics

Anna Puig

Objectius

- Es defineixen els principals elements gràfics d'una visualització.
- S'analitza el ***pipeline*** (o procés) de visualització de qualsevol aplicació gràfica. Es detallen les etapes implicades en el procés.
- S'estudien diferents ***arquitectures*** per implementar el *pipeline* de visualització: la clàssica i la centrada en la GPU.

NOTA: En algunes transparències hi trobareu citades les seccions del llibre de referència bàsic de l'assignatura que cal llegir per complementar els conceptes exposats.

[Angel2011] Edward Angel, Dave Shreiner, **Interactive Computer Graphics: A Top-Down Approach with Shader-Based OpenGL, 6/E**, ISBN-10: 0132545233. ISBN-13: 9780132545235, Addison-Wesley, 2011

Índex

2.1. Introducció

2.2. Elements gràfics

- Objectes
- Càmera
- Llums
- Escena

2.3. Pipeline gràfica

- Arquitectura clàssica
- Arquitectura GPU

Procés de visualització

Per a generar una imatge:

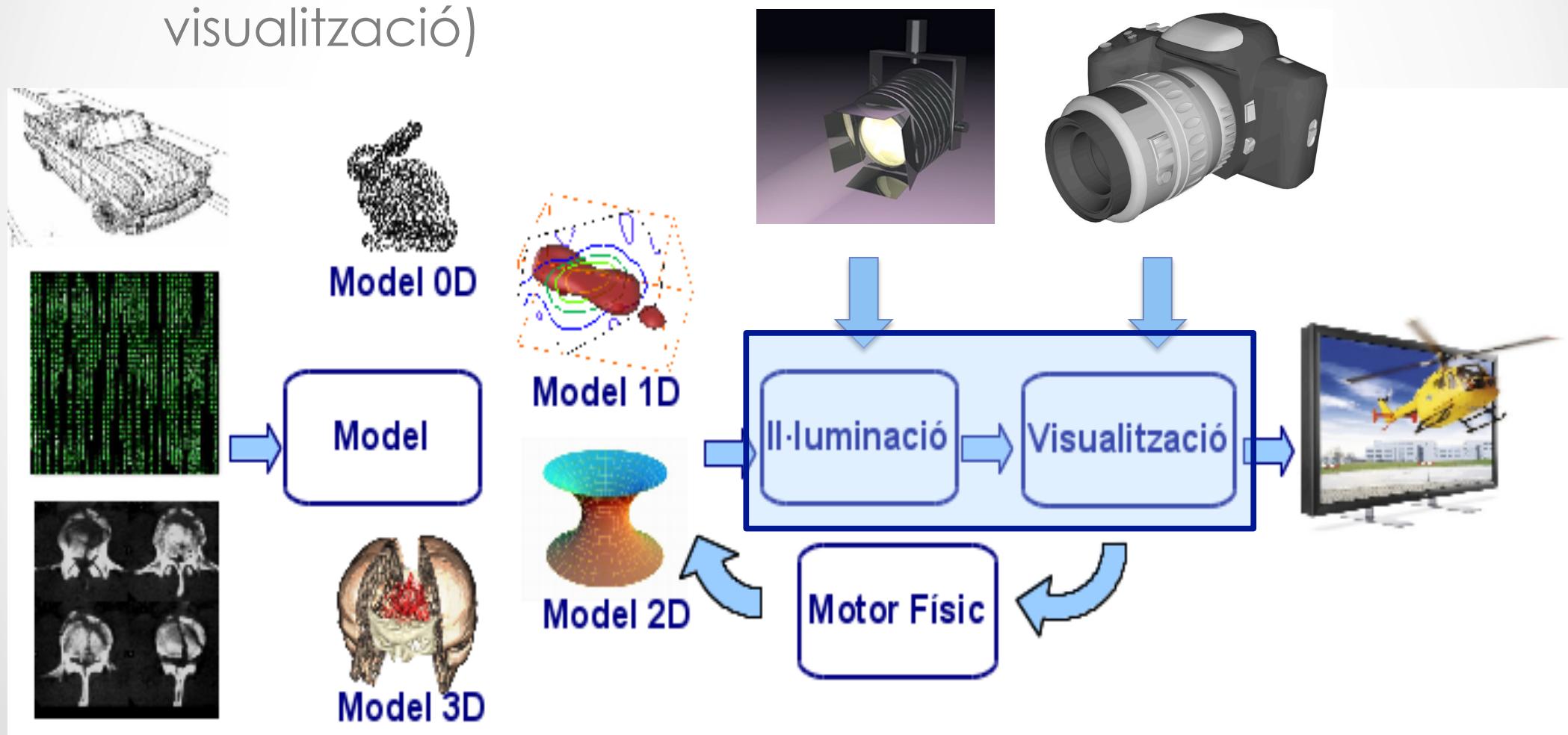


- Quins elements gràfics es necessiten?
- Quins passos es segueixen per generar-la? (pipeline de visualització)



Procés de visualització

El pipeline de visualització es centra en el procés d'il·luminació i visualització (paradigma modelatge-visualització)



Índex

2.1. Introducció

2.2. Elements gràfics (Modelat)

- Objectes
- Càmera
- Llums
- Escena

2.3. Pipeline gràfica

- Arquitectura clàssica
- Arquitectura GPU

Elements gràfics

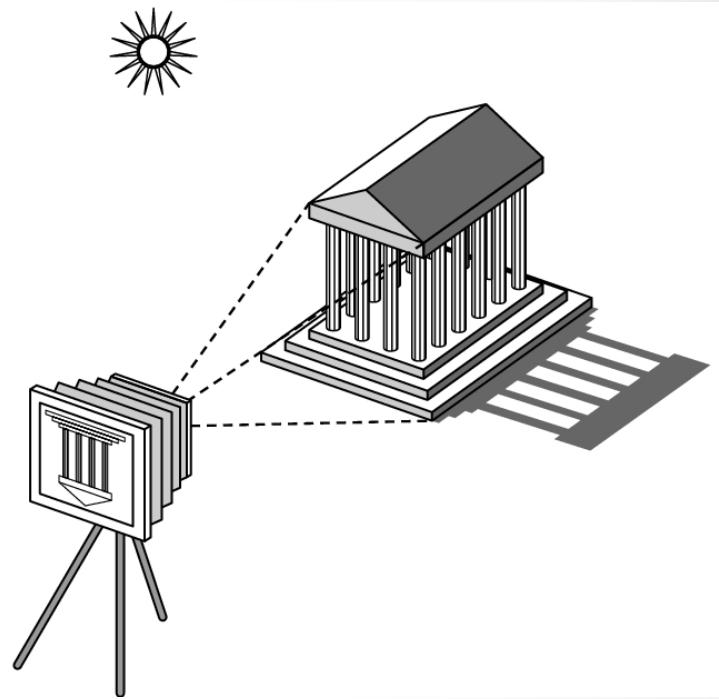
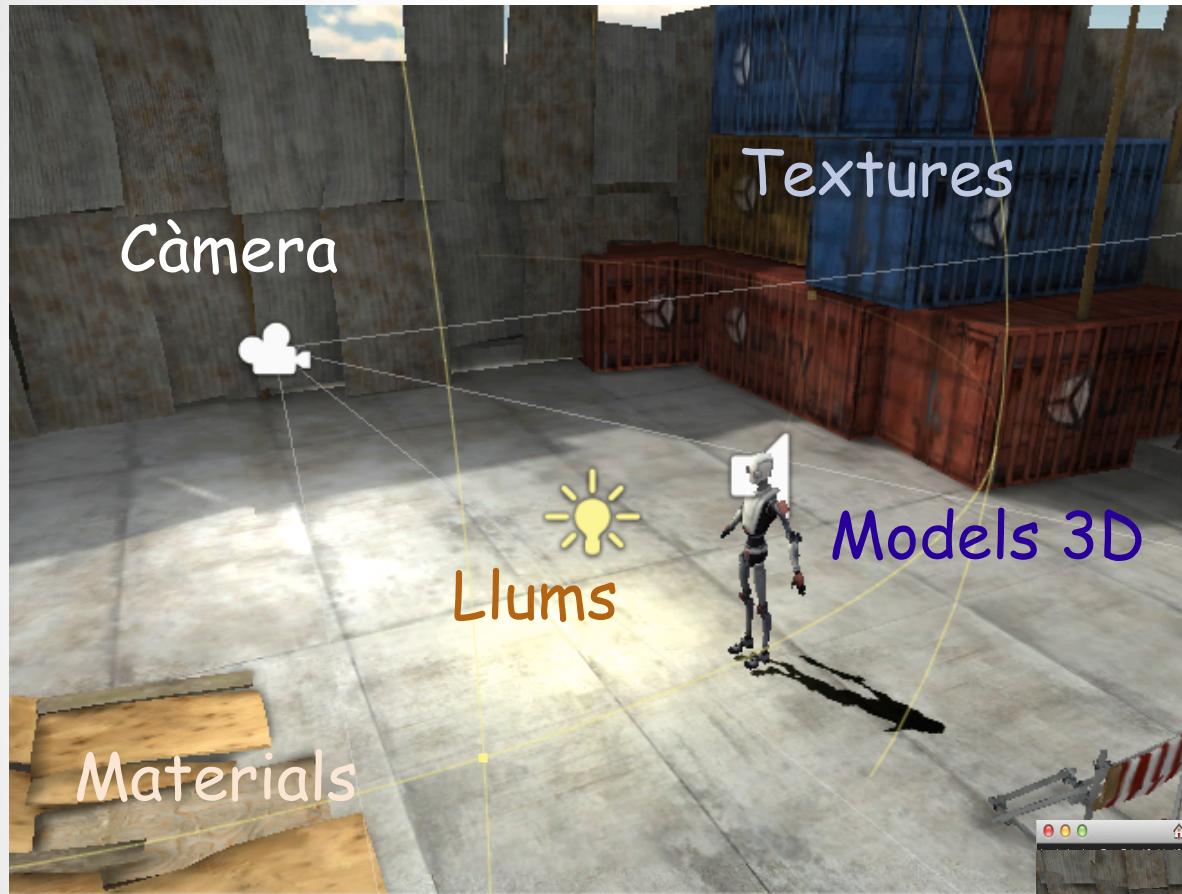
Quins elements gràfics es necessiten per a generar-la?

Els elements bàsics implicats en la visualització d'una escena són:

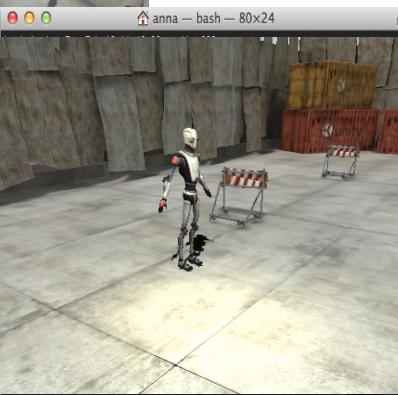
- Els **objectes** de l'escena en un sistema de coordenades (geometria, topologia i materials)
- La **càmera** o punt de vista des d'on s'observa l'escena
- La **lluminació** de l'escena (implica la definició de les fonts de llum de l'escena i el tipus de transport de la llum que es vol modelar en la imatge final)

Al final del procés s'obté una imatge 2D on cada píxel té associat un color (**frame buffer**)

Elements gràfics



Frame buffer



Elements gràfics: Models

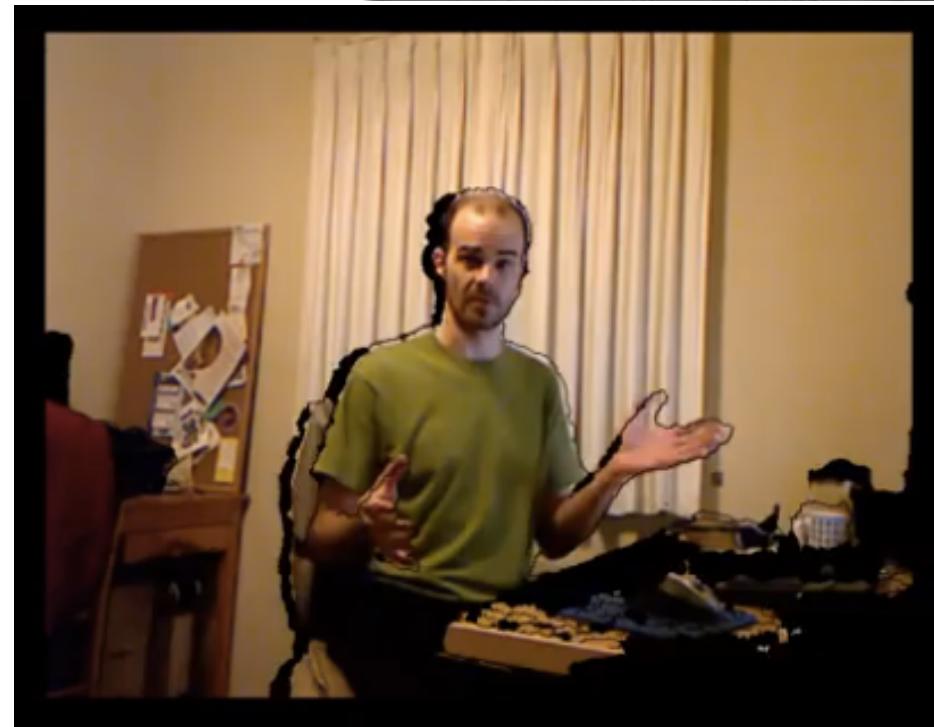
- Per **modelar** i definir els objectes:
 - Escanejat 3D
 - A partir d'aparells de captació de dades
 - Internament en el codi de l'aplicació (cub, esfera, objectes, fractal de Sierpiński, resultats de simulacions o mostrejos de funcions, etc): Modelat procedural, simulacions científiques....
 - Sistemes gràfics ja creats i exportar-los en formats estàndards (blender, Maya 3D, SolidWorks, 3DStudioMax, etc)
 - Dins del sistema de visualització com és el cas dels mons virtuals (Second-Life, OpenWonderland, Panda3D, google3D) o motors de jocs (unity3D)

Elements gràfics: Objectes

- Per **modelar** i definir els objectes:
 - Escanejat 3D



<http://www.david-3d.com>



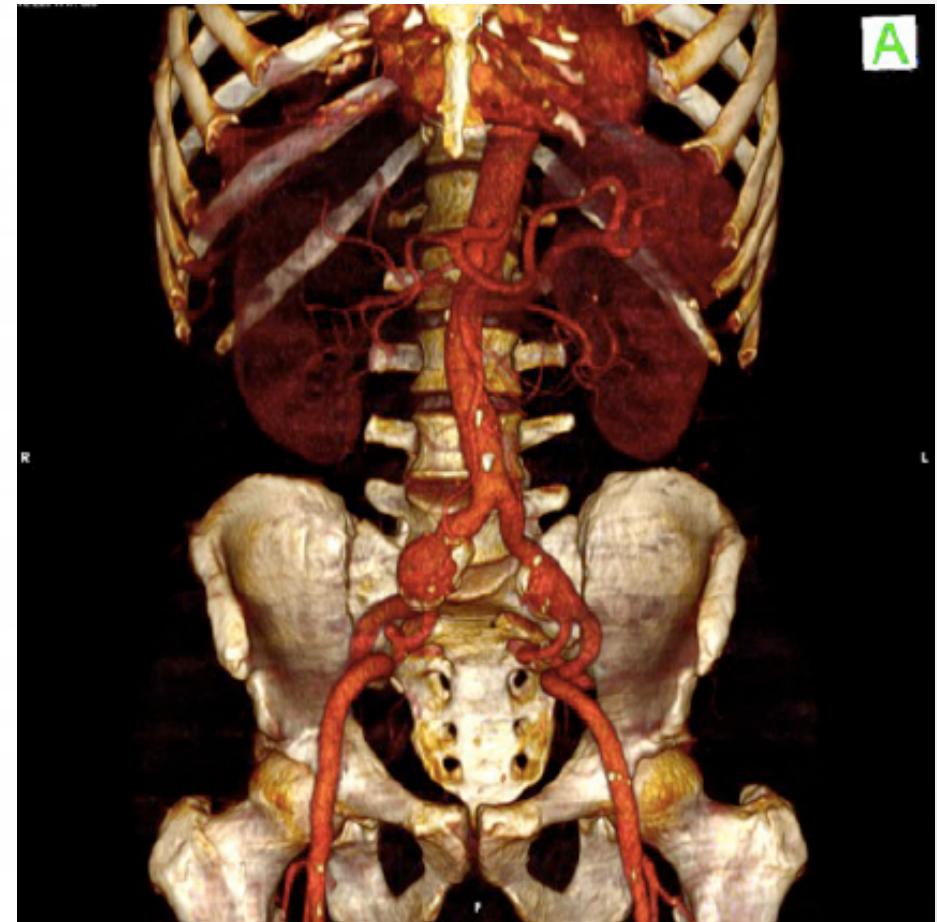
<http://www.youtube.com/watch?v=7QrnwoO1-8A>

Elements gràfics: Objectes

- Per **modelar** i definir els objectes:
 - A partir d'aparells de captació de dades



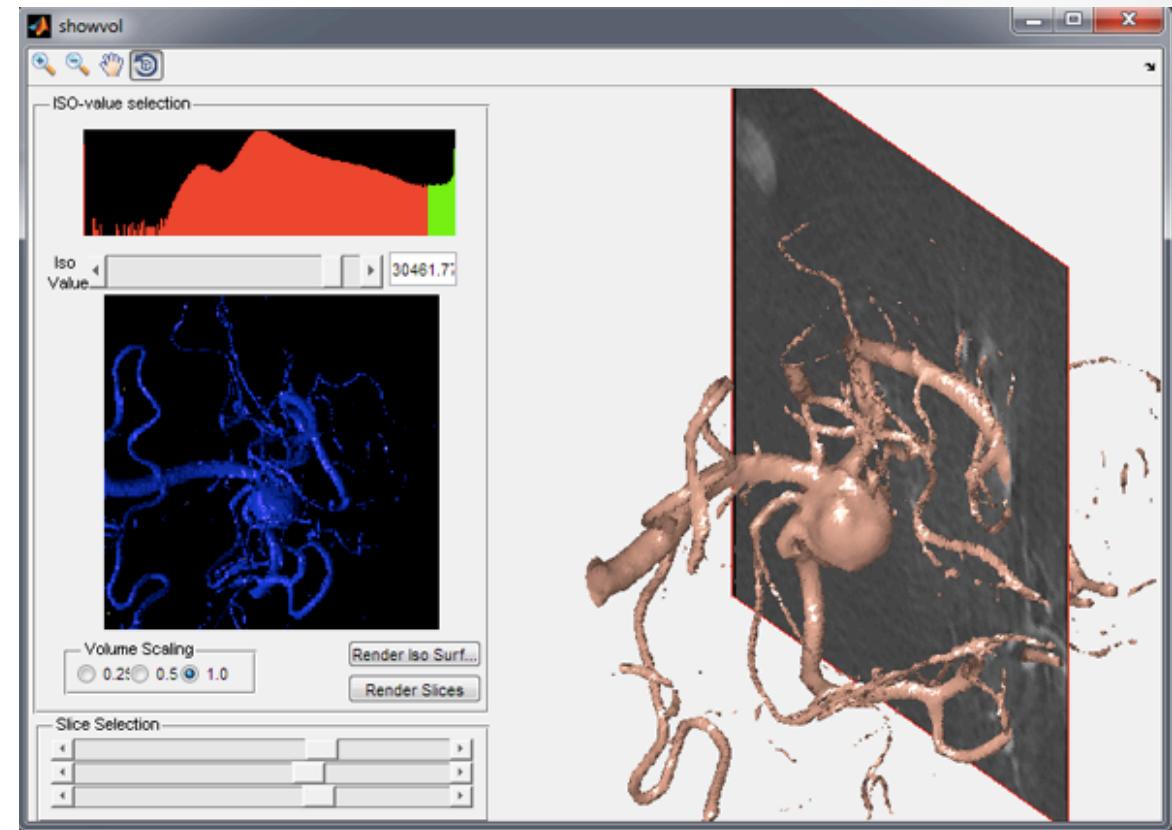
<http://www.alma-medical.com/portfolio/alma-iclinic/>



OsiriX Imaging Software and Imaging Economics

Elements gràfics: Objectes

- Per **modelar** i definir els objectes:
 - Models procedimentals
 - Extracció de superfícies

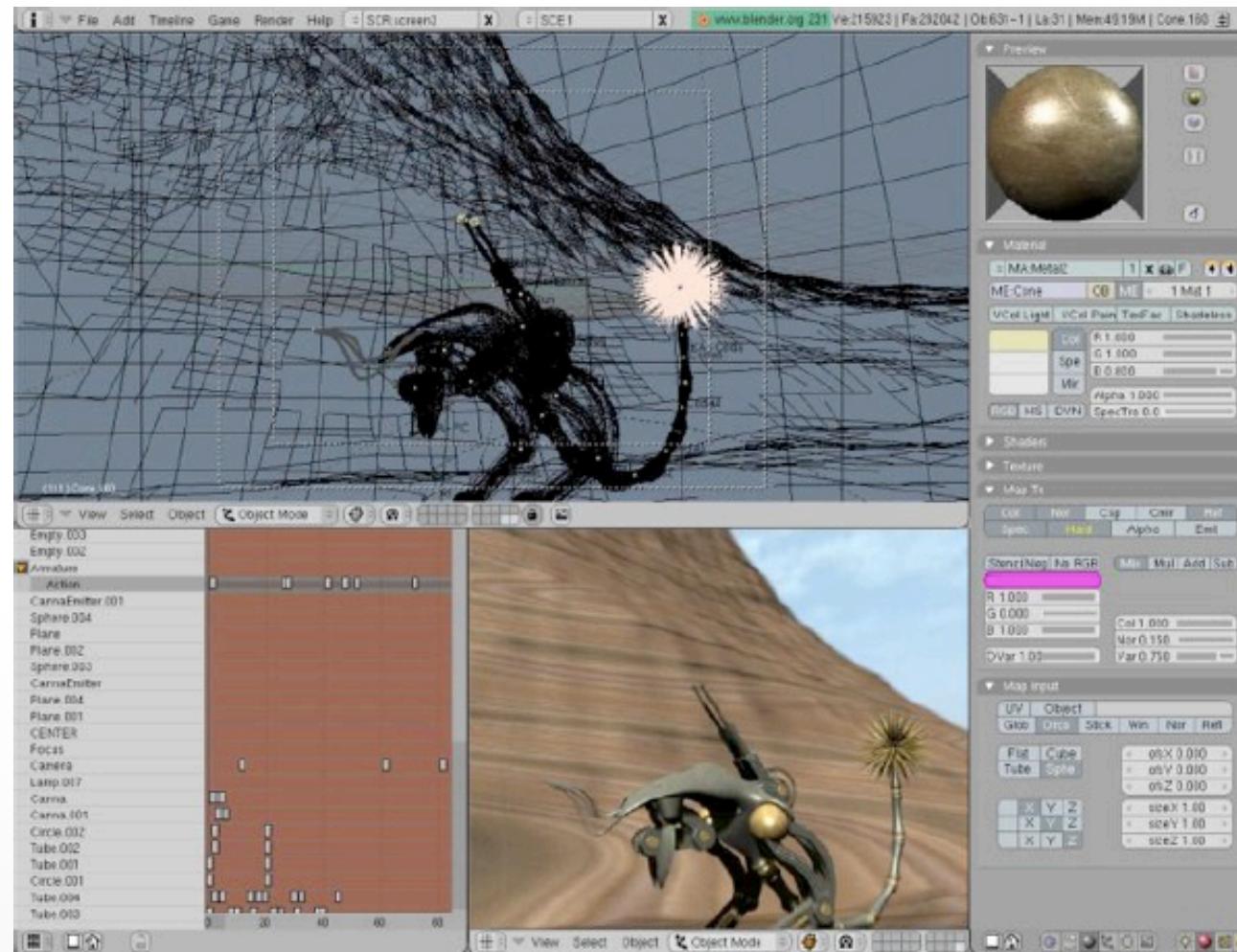


<http://www.mathworks.com/matlabcentral/fileexchange/25987-showvol-isosurface-render>

Terragen 3 gallery: <http://planetside.co.uk/products/terragen3>

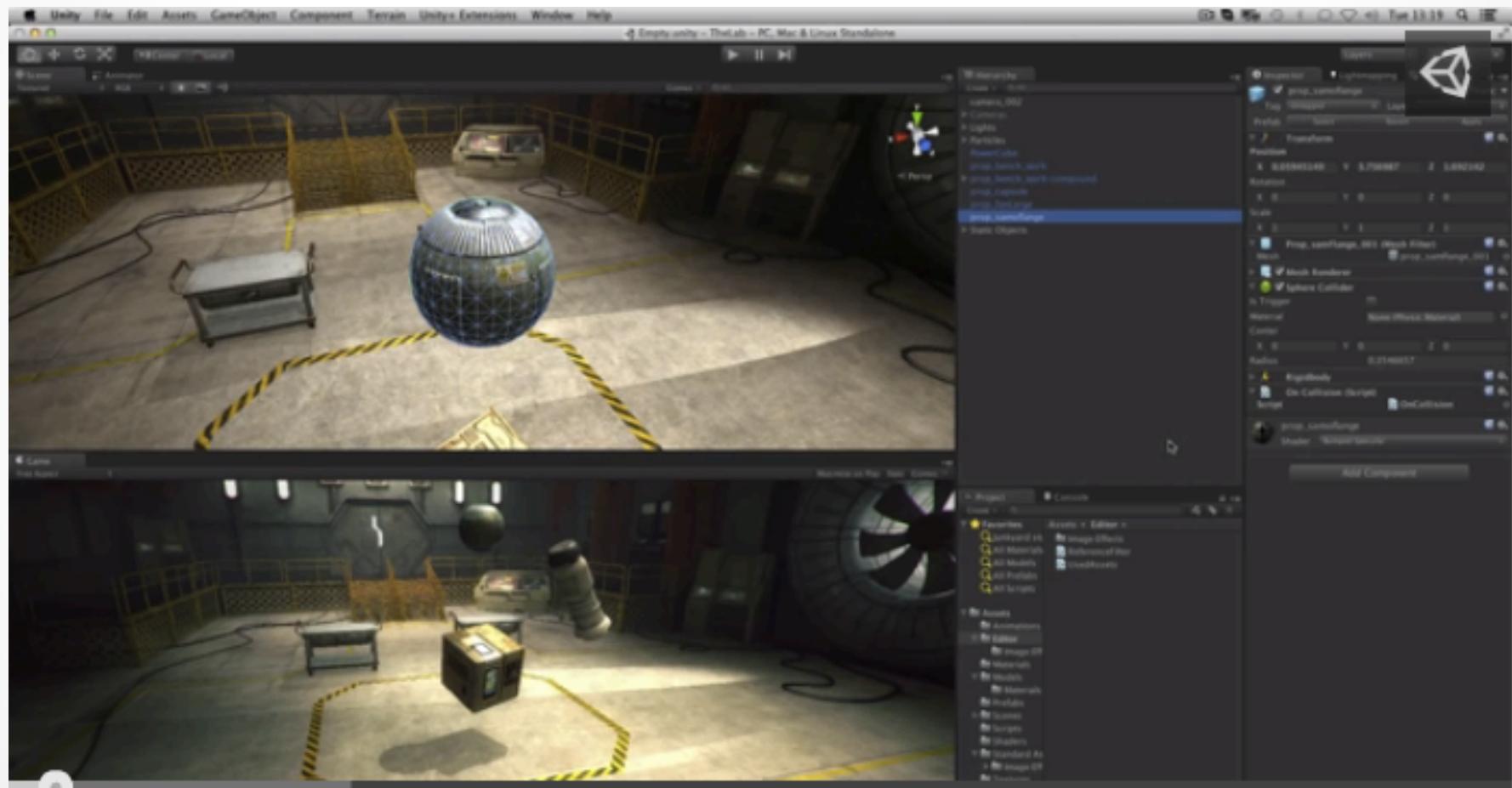
Elements gràfics: Objectes

- Per **modelar** i definir els objectes:
 - Aplicatius de modelat d'escenes



Elements gràfics: Objectes

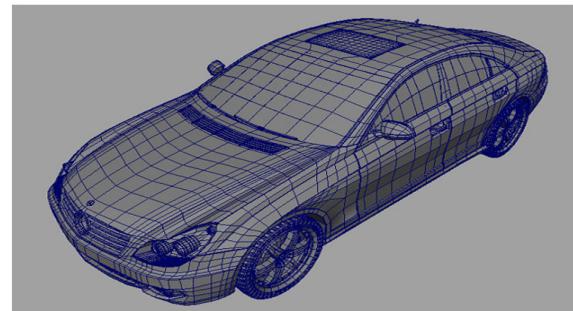
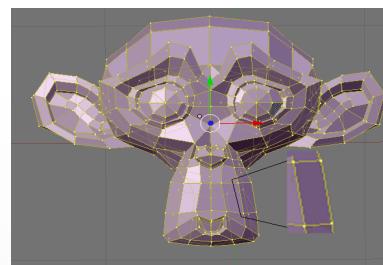
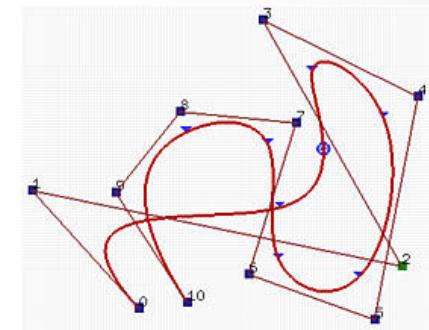
- Per **modelar** i definir els objectes:
 - Motors gràfics i mons virtuals



Elements gràfics: Objectes

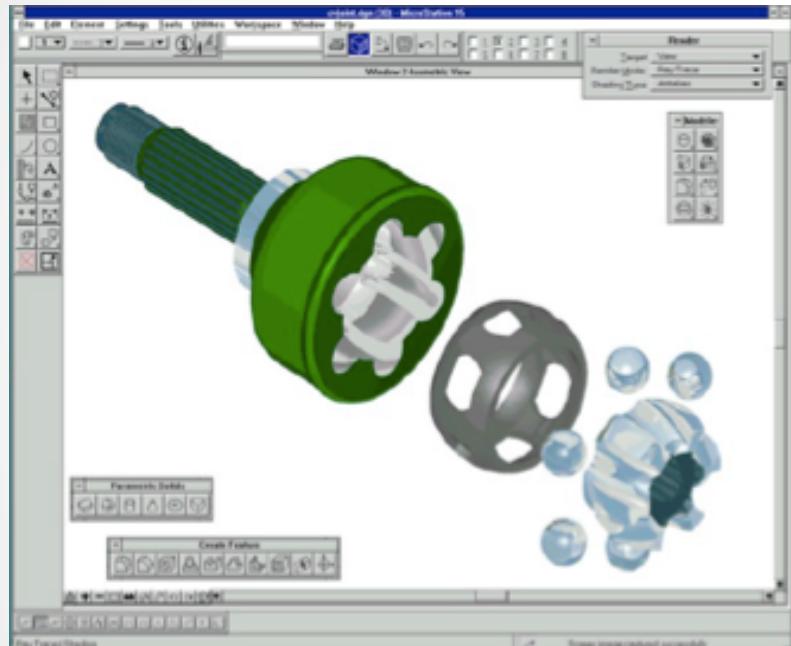
Els objectes que formen una escena poden ser:

- **Objectes superficials:** només representen la frontera de l'objecte amb l'exterior. Poden ser de diferents tipus:
 - Sòlids (esferes, cons, cilindres, etc.)
 - Malles poligonals i superfícies suaus (mesh)
 - Punts
 - Blobby models (models deformables)
- Mètodes de modelatge: subdivisió, operacions booleanes, edició de corbes i superfícies

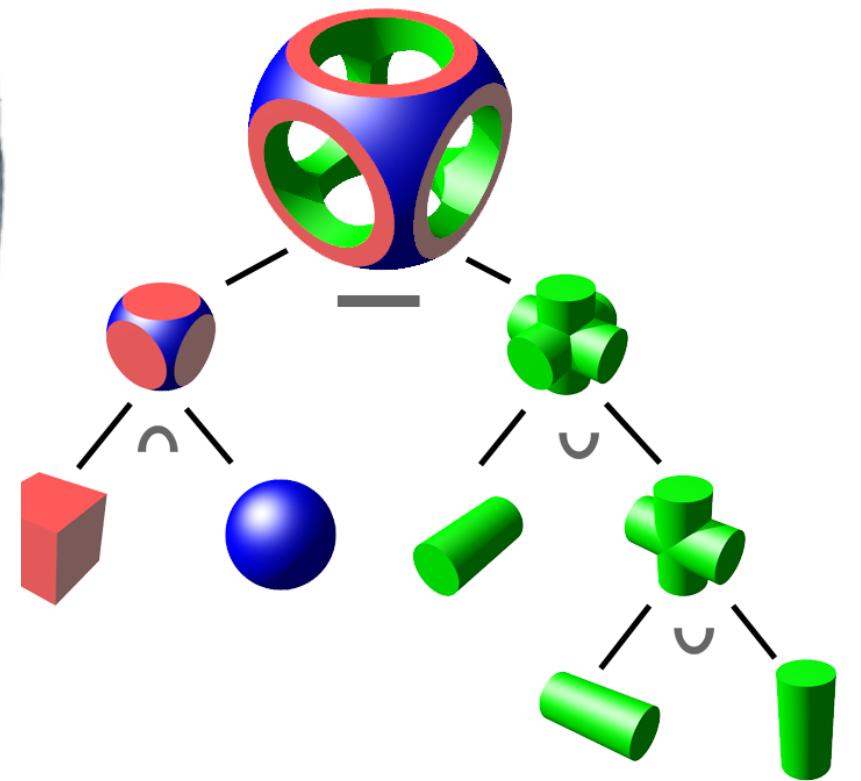
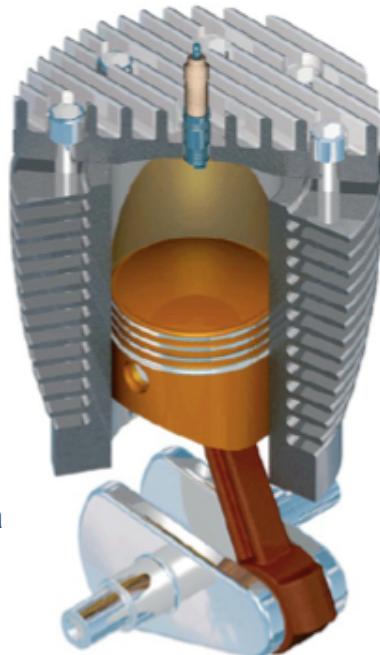


Elements gràfics: Objectes

- **Sòlids:** basats en objectes paramètrics com esferes, cilindres i cons, amb operacions booleanes (arbres CSG)



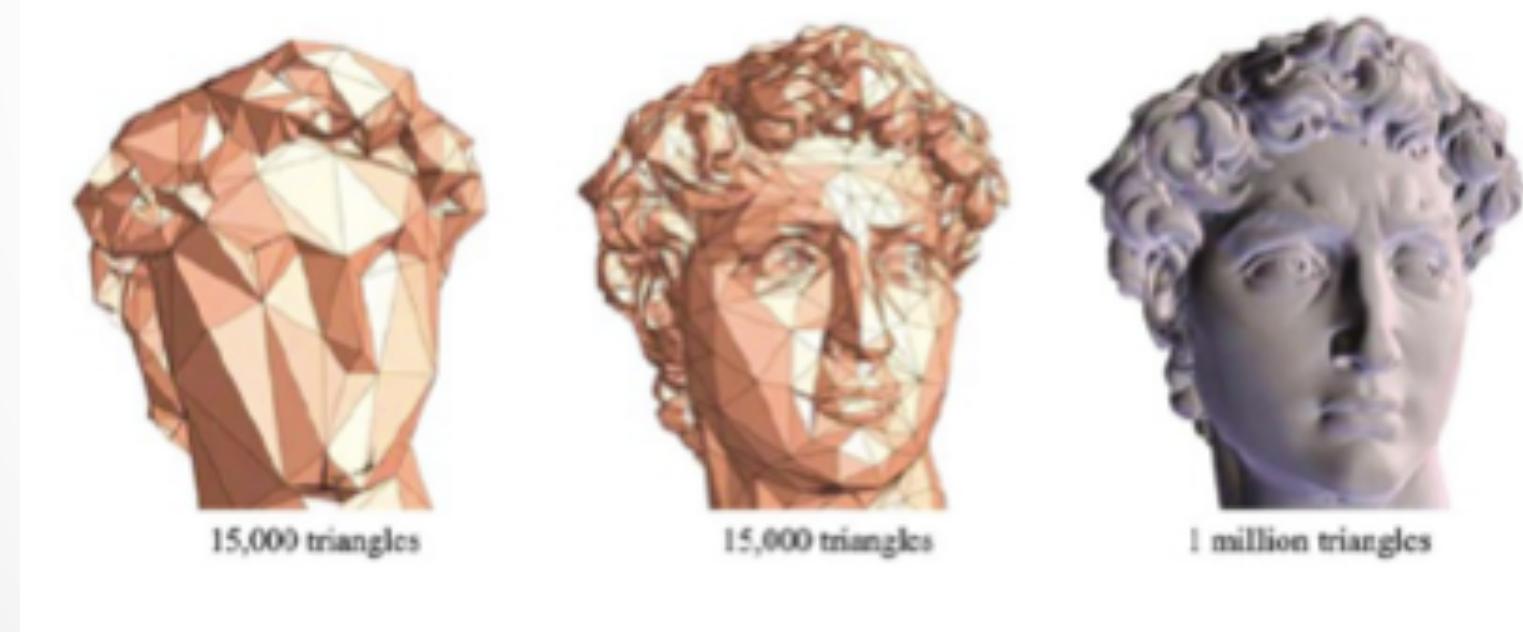
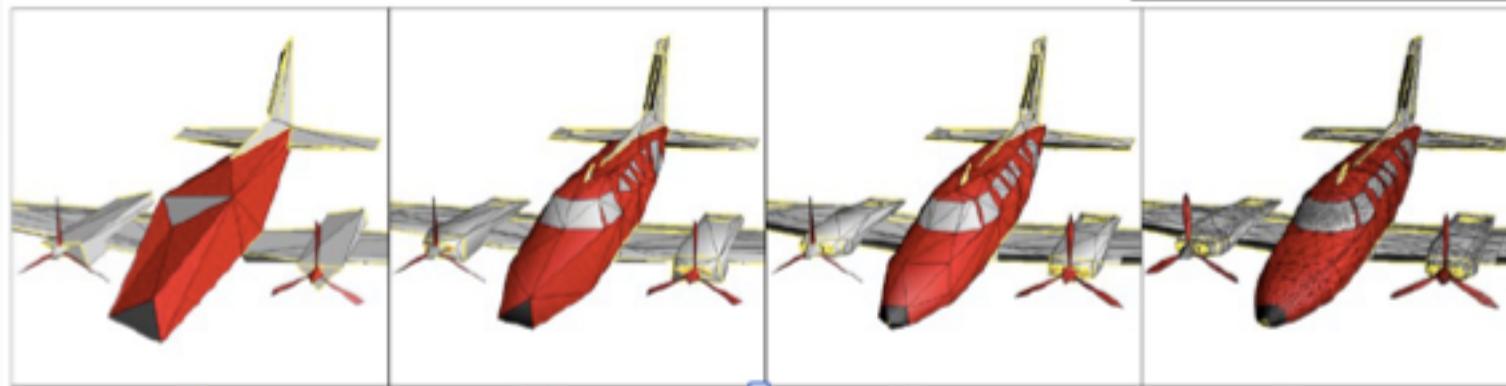
Computer Desktop Encyclopedia



<http://freesdk.crydev.net/display/SDKDOC2/Solid+Tool>

Elements gràfics: Objectes

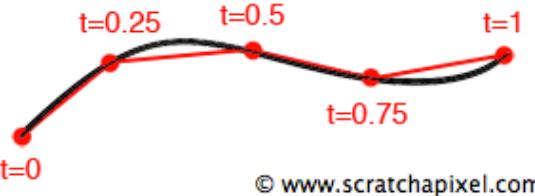
- **Malles poligonals:**



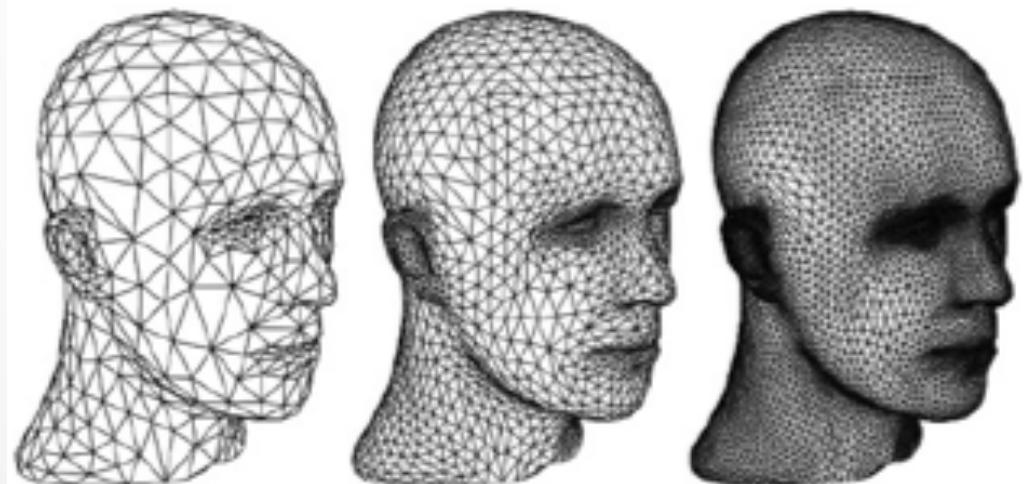
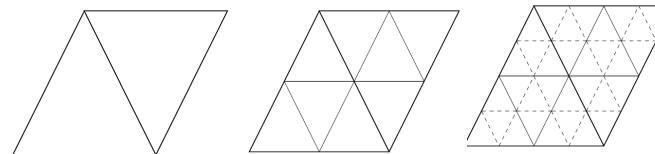
Hoppe et al., SIGGRAPH 1993 i 1994, TOG 2004

Elements gràfics: Objectes

- **Superfícies suaus:** basades en Bézier, superfícies bicubiques Splines, etc.
 - Corbes de Bézier
 - Subdivisió recursiva



© www.scratchapixel.com

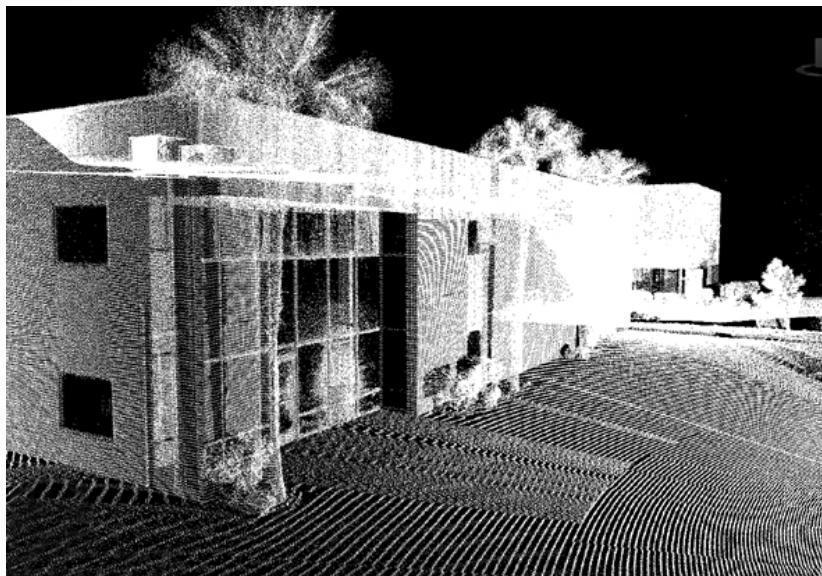


Hoppe et al., SIGGRAPH 1993 i 1994, TOG 2004



Elements gràfics: Objectes

- Punts i normals: (point cloud)



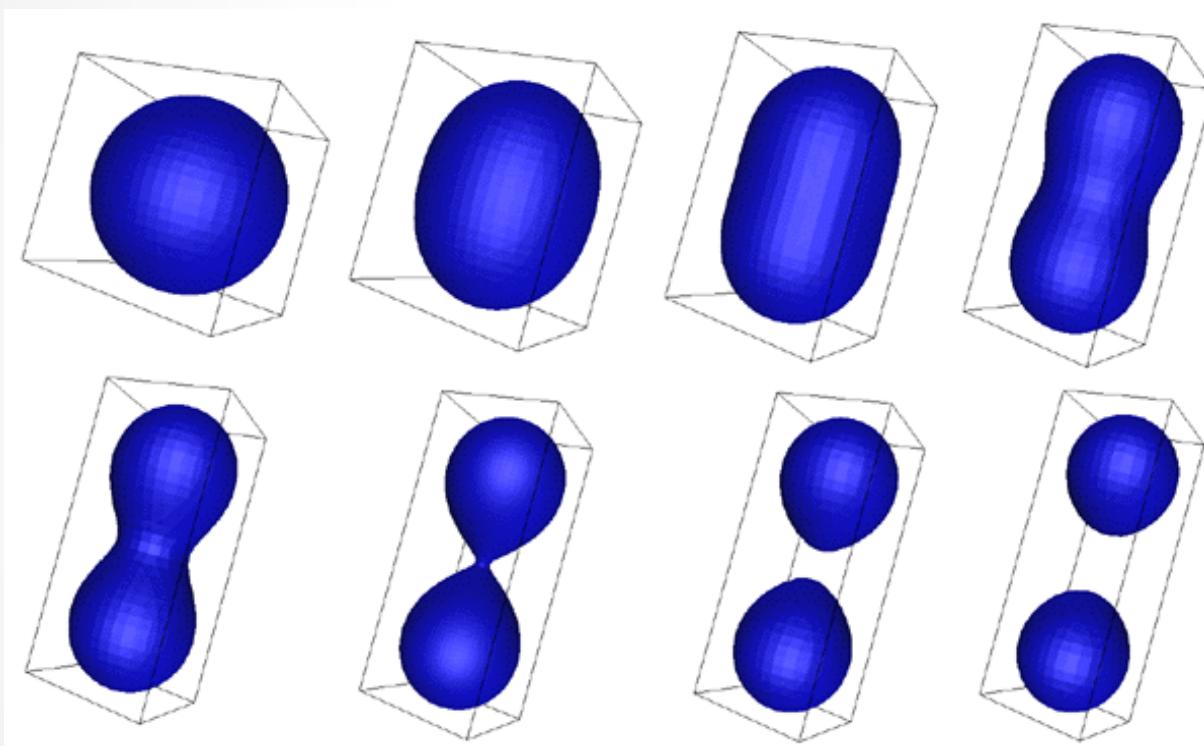
http://revittunes.com/gallerybg.php?img=revit-Leica_Point_Cloud_Scanning.jpg&t=0&publici=18



Zwicker et al. I Pauly , Gross, SIGGRAPH, 2001
Semestre Primavera 2014 ● 19

Elements gràfics: Objectes

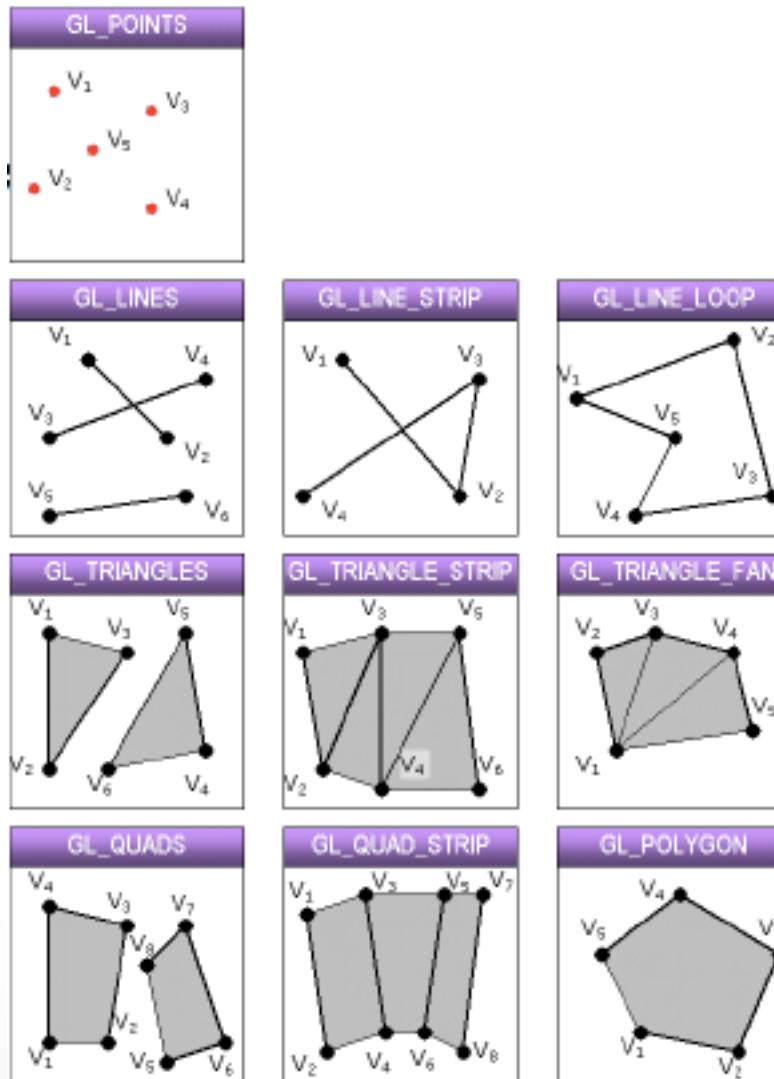
- **Blobby models**



Brian Wyvill, Craig McPheeers, Geoff Wyvill
Soft Objects
Advanced Computer Graphics (Proc. CG Tokyo 1986)

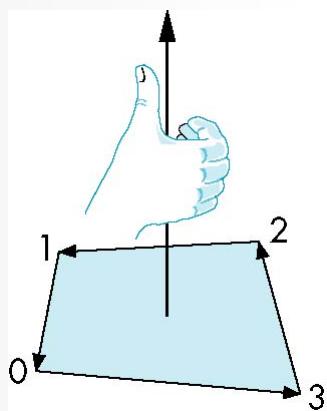
Elements gràfics: Objectes

- Representació de *mesh* de polígons:
 - En GL es tenen diferents maneres de pintar

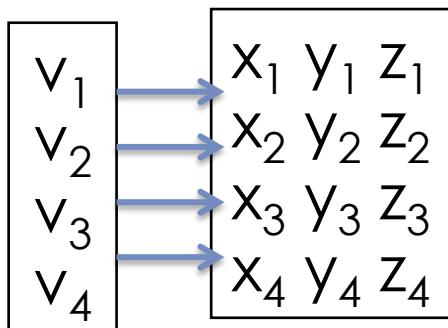


Elements gràfics: Objectes

- Representació de mesh de polígons:
 - Llista de vèrtexs



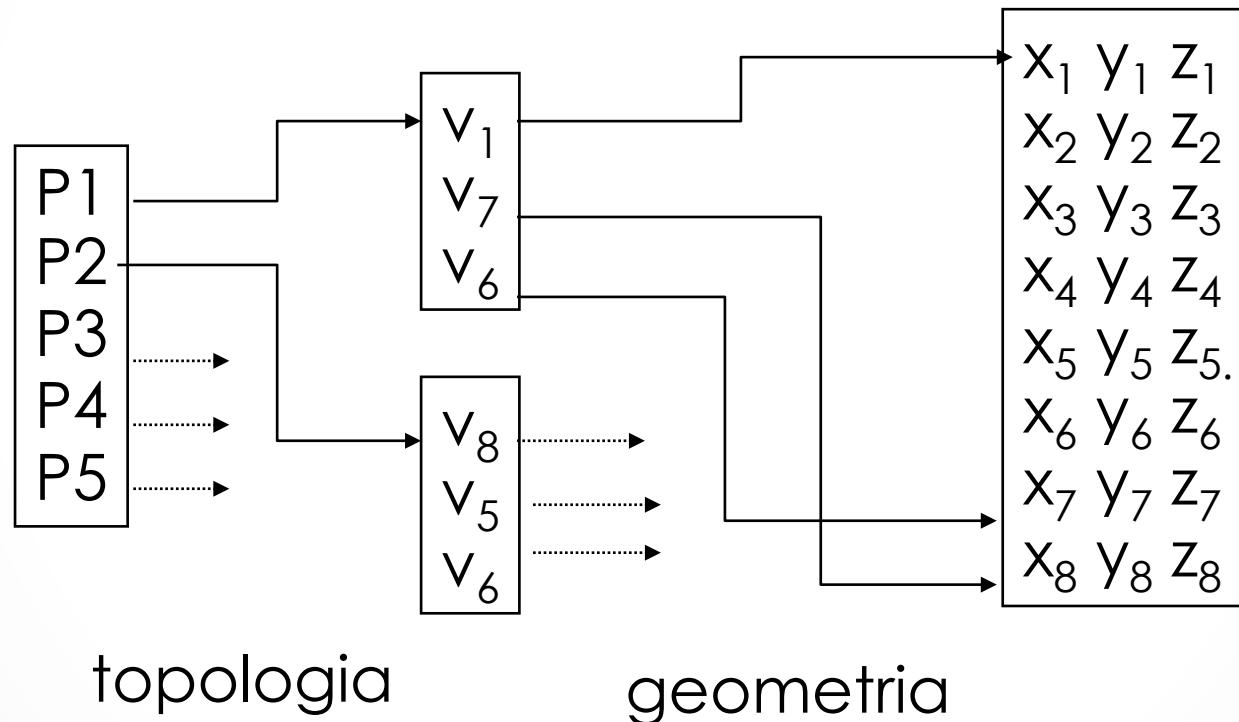
Normal



```
float v1[3] = {-1.0,-1.0,0.0};  
float v2[3] = { 1.0,-1.0,0.0};  
float v3[3] = { 1.0, 1.0,0.0};  
float v4[3] = {-1.0, 1.0,0.0};  
  
glBegin(GL_POLYGON);  
    glVertex3fv(v1);  
    glVertex3fv(v2);  
    glVertex3fv(v3);  
    glVertex3fv(v4);  
glEnd();
```

Elements gràfics: Objectes

- Representació de mesh de polígons:
 - *Llistes de polígons + vèrtexs*



Elements gràfics: Objectes

- Representació de mesh de polígons:

- *Llistes de polígons + vèrtexs*

```
typedef float Point[3];
```

```
Point verts[8] = {
```

```
    {-1.,-1.,-1.},
```

```
    { 1.,-1.,-1.},
```

```
    { 1., 1.,-1.},
```

```
    {-1., 1.,-1.},
```

```
    {-1.,-1., 1.},
```

```
    { 1.,-1., 1.},
```

```
    { 1., 1., 1.},
```

```
    {-1., 1., 1.},
```

```
};
```

```
face(int a, int b, int c, int d) {
```

```
    glBegin(GL_POLYGON);
```

```
    glVertex3fv(verts[a]);
```

```
    glVertex3fv(verts[b]);
```

```
    glVertex3fv(verts[c]);
```

```
    glVertex3fv(verts[d]);
```

```
    glEnd();
```

```
}
```

```
// Note consistent ccw orientation!
```

```
cube() {
```

```
    face(0,3,2,1);
```

```
    face(2,3,7,6);
```

```
    face(0,4,7,3);
```

```
    face(1,2,6,5);
```

```
    face(4,5,6,7);
```

```
    face(0,1,5,4);
```

```
}
```

Elements gràfics: Objectes

- Representació de mesh de polígons:
 - Llistes de **cares+arestes+vèrtexs**:
 - Winged-edge (optimització)

<http://www.baumgart.org/winged-edge/winged-edge.html>

Cara	Aristas				
1	0	1	2	3	NULL
2	4	5	6	1	NULL
3					

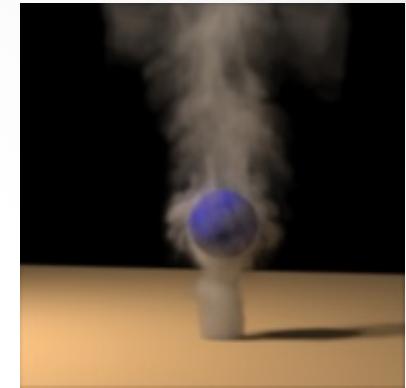
Aristas	V _i	V _f	C _{izq}	C _{der}
0	0	1	1	3
1	1	2	1	2
2	2	3	1	4
3	3	0	1	5
4	1	5	2	3
5	5	4	2	6
6	4	2	2	4

Vèrtice	X	Y	Z
0	0	0	0
1	0	2	-1
2	0	3	2
3	0	1	2
4	2	4	1
5	2	3	-1

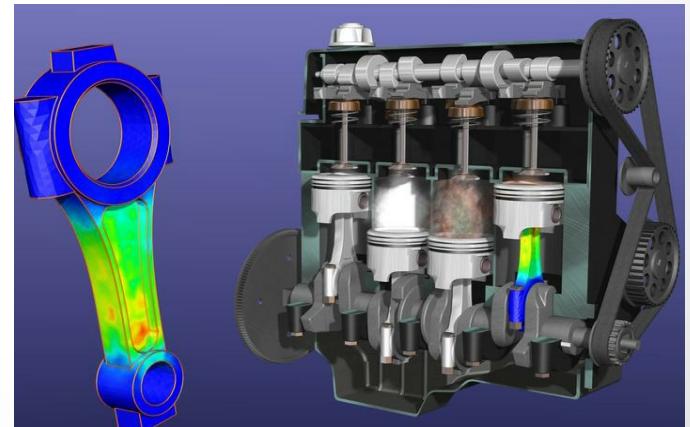
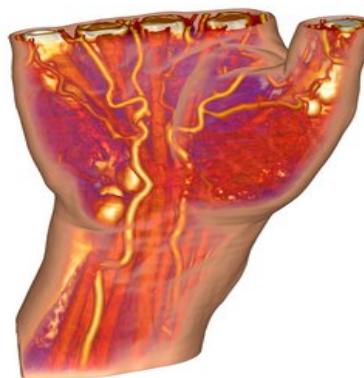
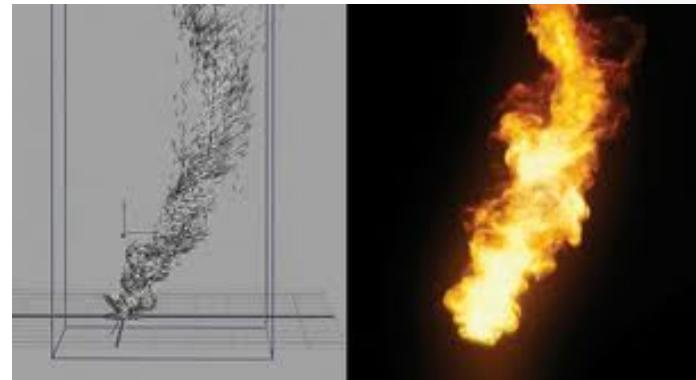
Elements gràfics: Objectes

Els objectes que formen una escena poden ser:

- **Objectes volumètrics:** es representa el volum intern dels objectes o característiques de l'espai com temperatura, densitat, etc.):



- Basats en punts (partícules)
- Vòxels (o píxels 3D)
- Tetràedres

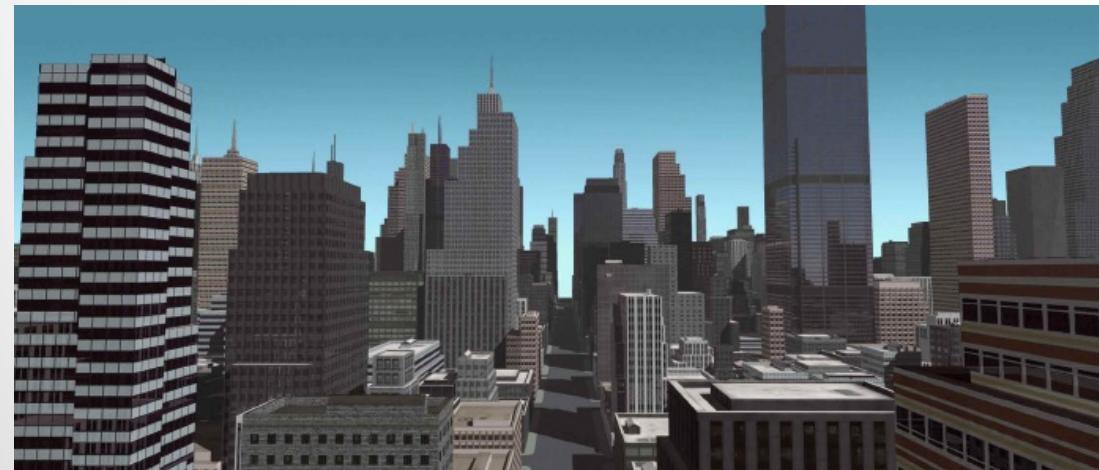
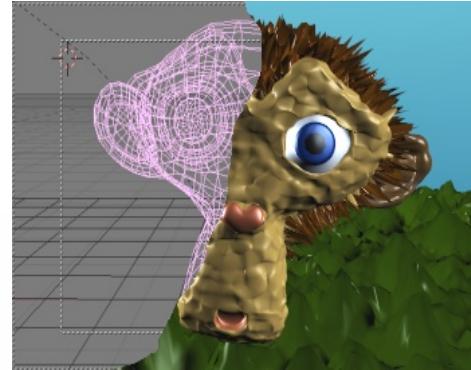


Es tractaran en el Tema 6 de l'assignatura

Elements gràfics: Objectes

El **realisme** dels objectes visualitzats implica detallar petites característiques que poden ser:

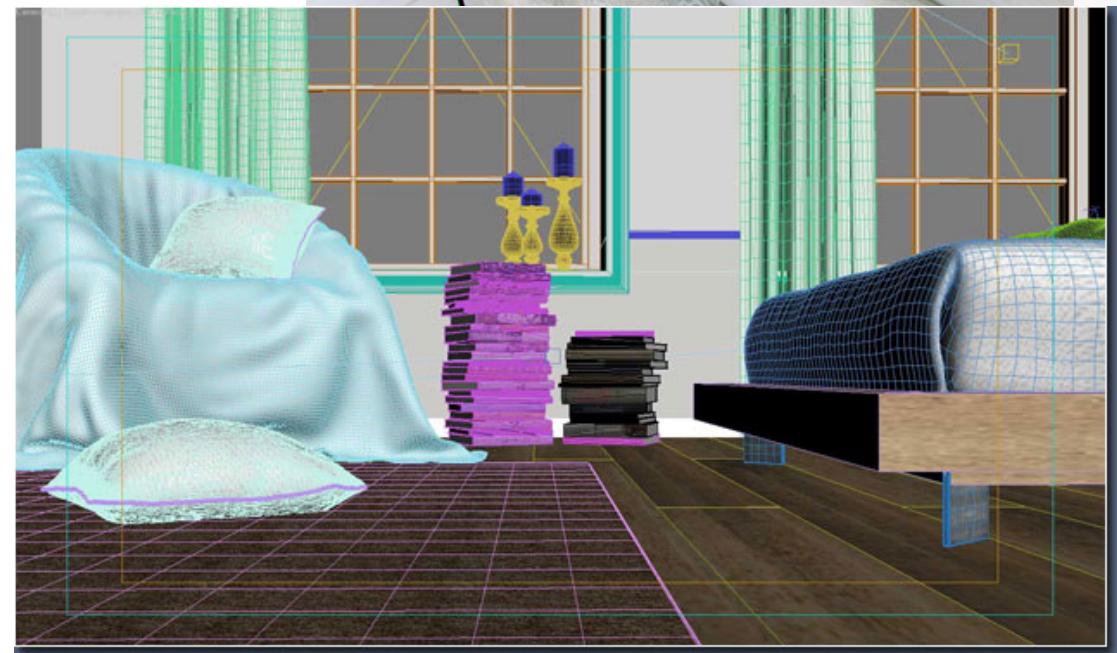
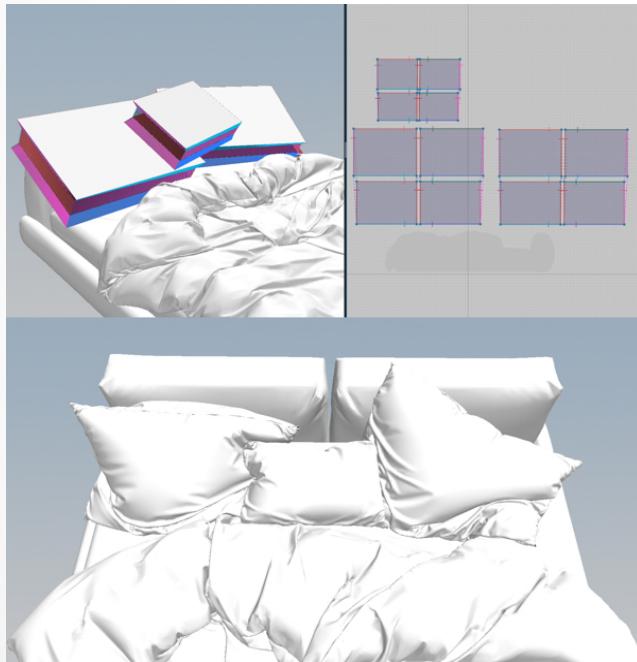
- **Modelades amb punts addicionals**
- **Definint materials** (Color difús Od (R, G, B), opacitat, color especular Os(R, G, B), ...) i **textures**



Es tractaran en el Tema 5 de l'assignatura

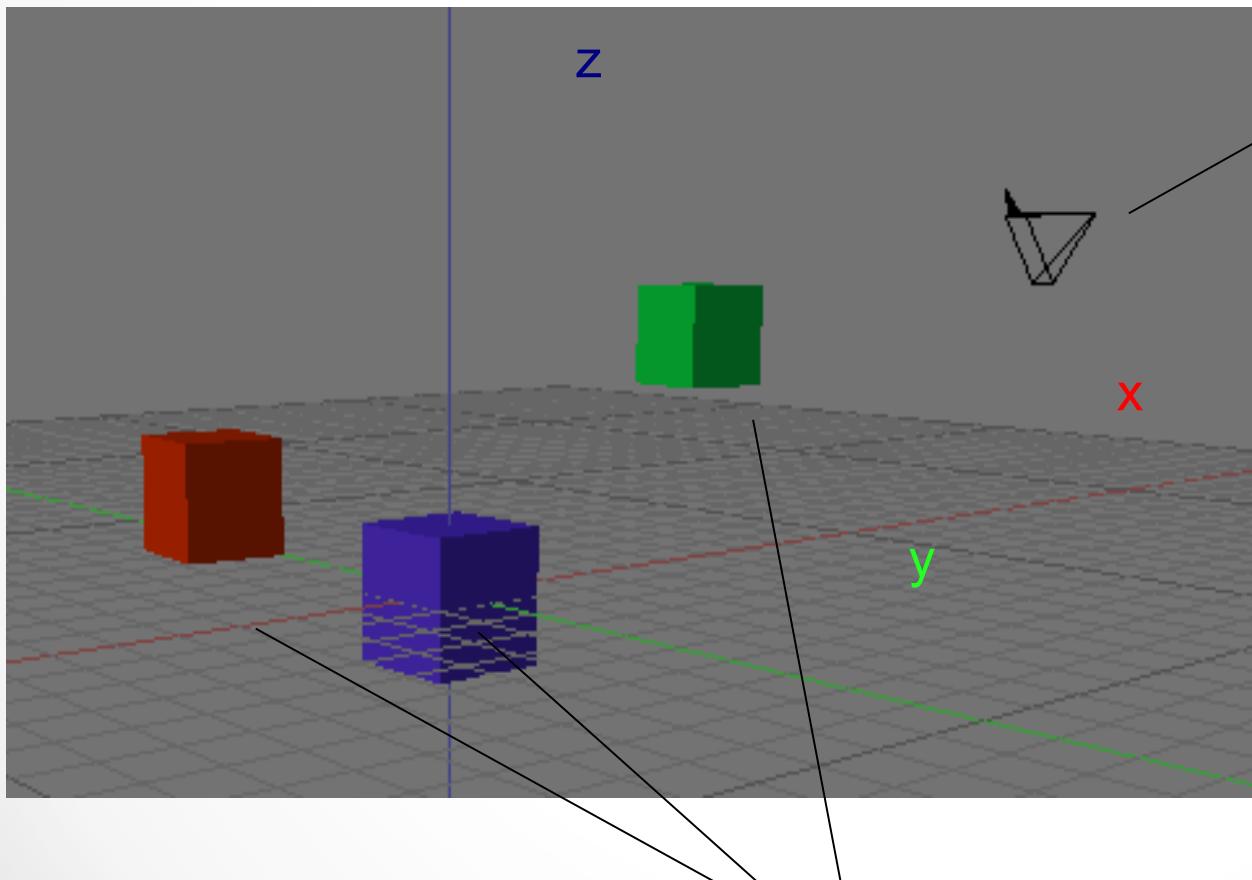
Elements gràfics: Objectes

- Informació bàsica:
 - Llista de vèrtexs (posició, normal, material i textura)
 - Llista de cares (punters als vèrtexs)



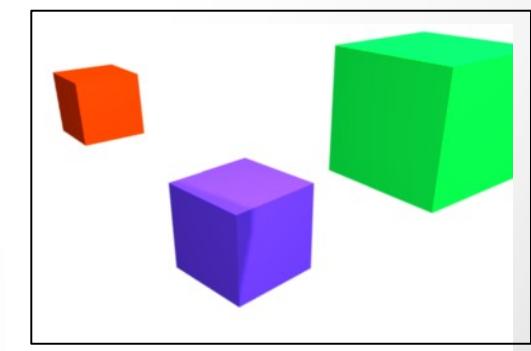
Elements gràfics: Càmera

- Com es troba la visualització final dels objectes definits? Cal definir la **càmera** on es situa l'observador



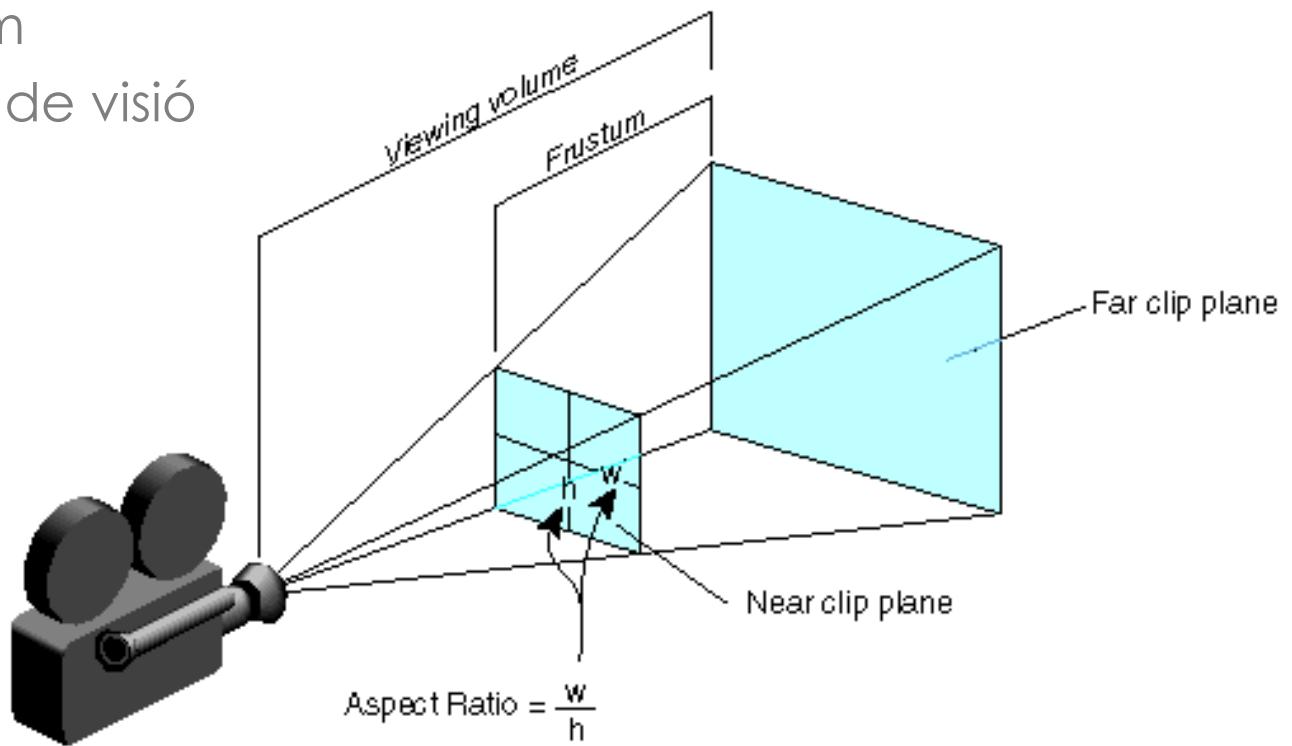
objectes

[1] Veure secció 1.5 del llibre [Angel2011]



Elements gràfics: Càmera

- Components de la càmera:
 - Posició de l'observador
 - VRP: view reference point
 - VUP: vector de verticalitat
 - D: distància de l'observador al VRP
 - Fustrum
 - Volum de visió

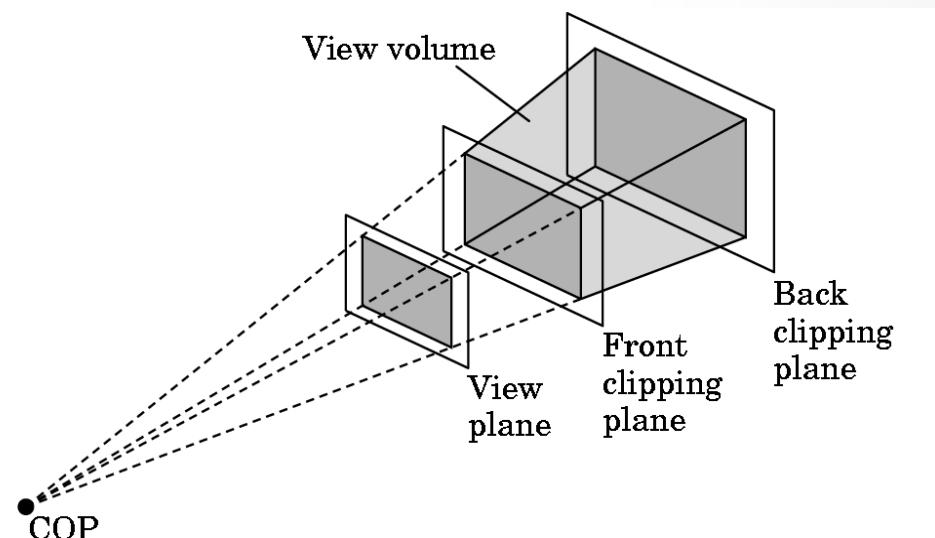
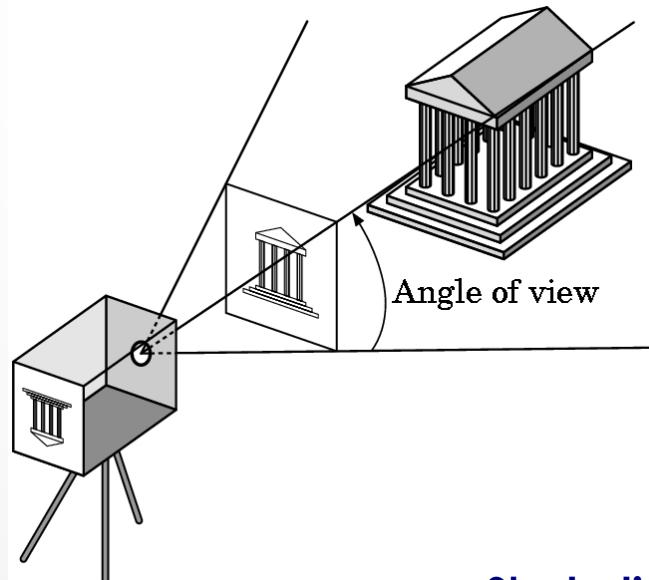


Elements gràfics: Càmera

Clipping (o retallat): La càmera només pot veure una part del món o de l'escena:

- Els objectes que no estan dins d'aquest volum (o *frustum*) es diu que son retallats (*clipped out*) de l'escena.

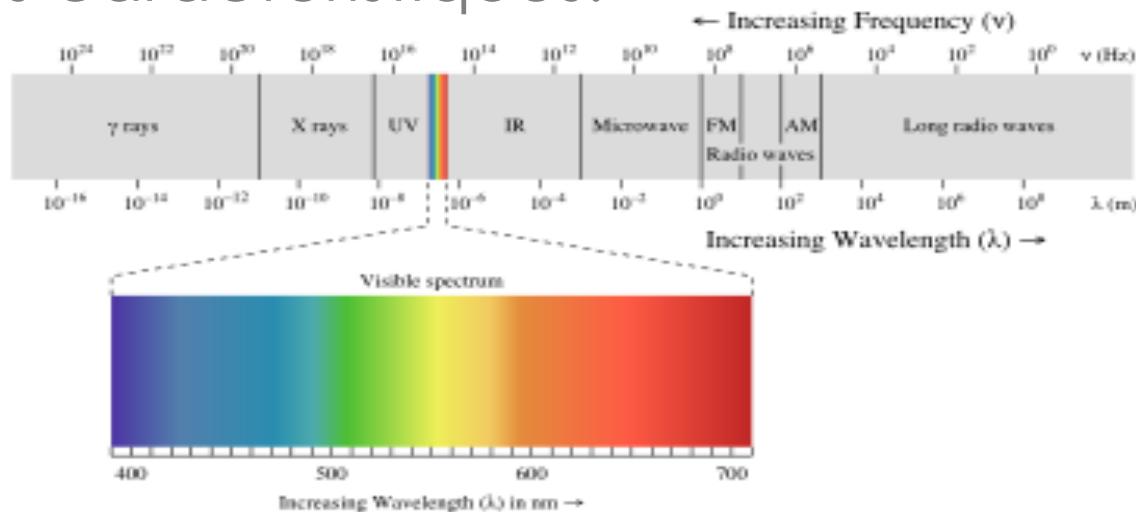
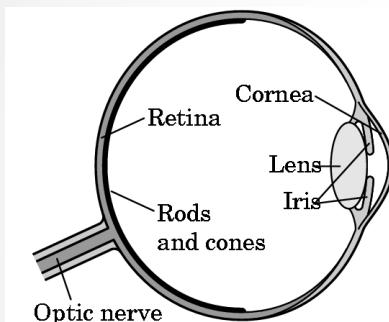
Pla de projecció (view plane): és el pla on es projecta l'escena (window) donant el que es veurà en el *frame buffer* final (o *viewport*)



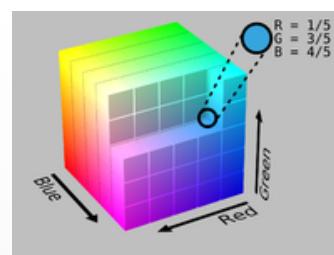
S'estudiaran en el Tema 4

Elements gràfics: Llum

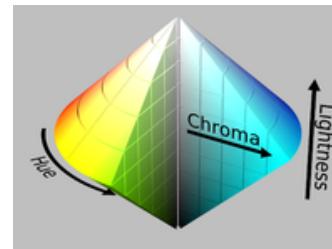
- El Color percutut per l'ull humà (cons i cilindres) té diferents característiques.



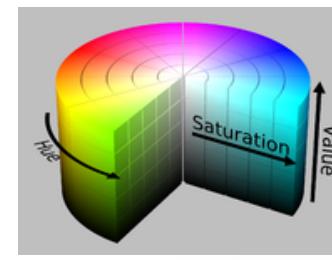
- Intensitat, saturació i brillantor (hue, saturation and lightness): diferents codificacions



RGB



HLS

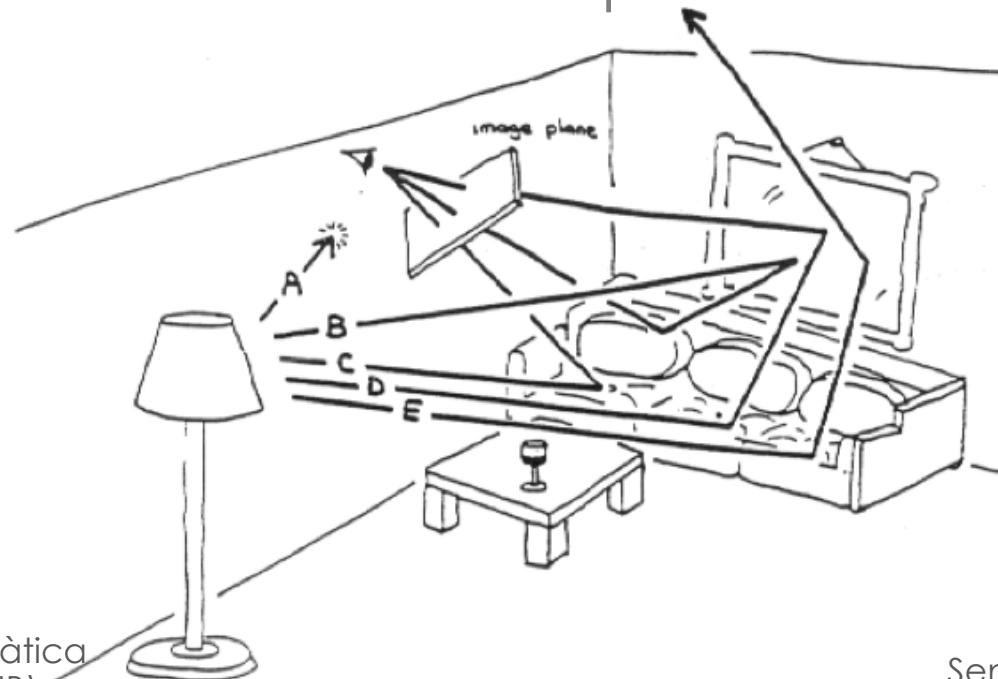


HSV

[1] Veure secció 1.3.2 del llibre [Angel2011]

Elements gràfics: Llum

- Les **fons de llum** són els emissors de llum.
- Model geomètric de la llum és un conjunt de línies rectes.
- Una font de llum ideal emet energia des d'un punt a una o més freqüències cap a totes les direccions
- Simulació del model de transport de la llum



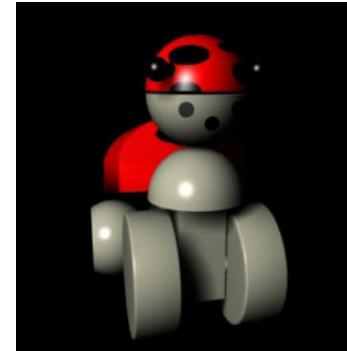
Elements gràfics: Llum

En una escena, normalment es defineix:

- **Color de background:** en general no contribueix a la il·luminació, excepte en visualitzacions molt realistes.

- **Focus de llum externs:**

- Intensitat de 0 a 1 en RGB
- Tipus:
 - Puntuals: (x, y, z)
 - Direccionals: (vx, vy, vz)
 - Spots: posició + direcció + angle sòlid angle

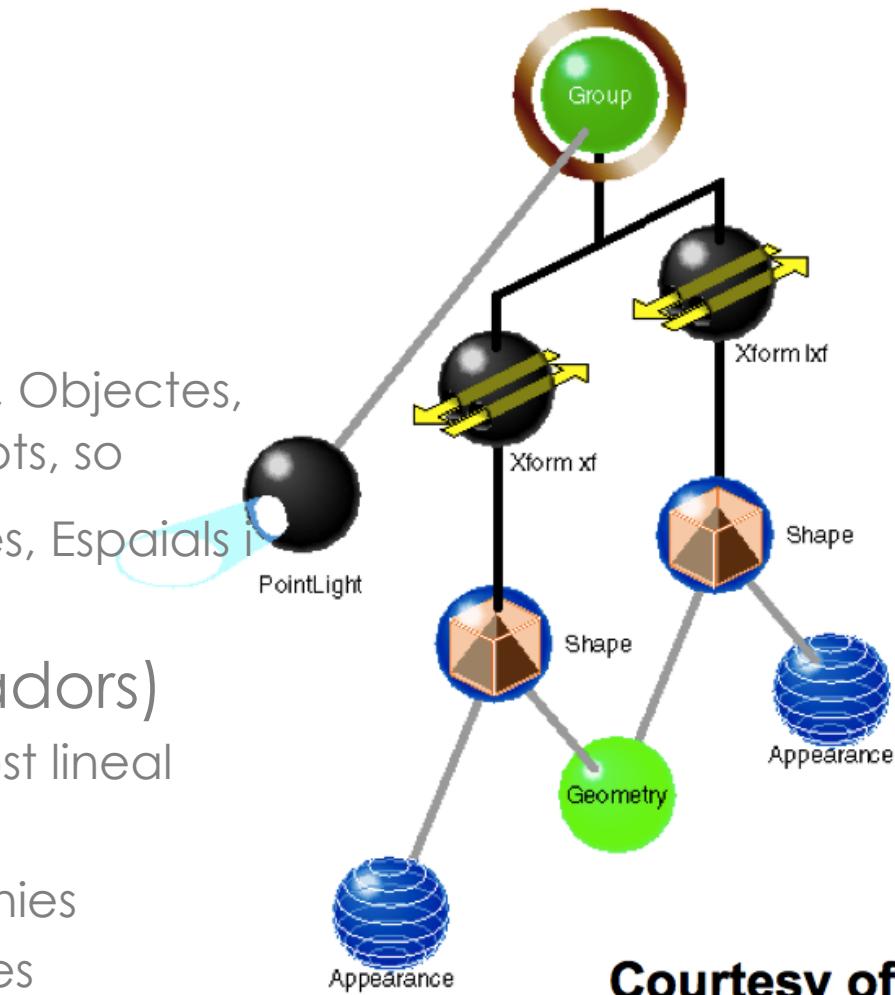


- **Llums internes:** objectes emissors



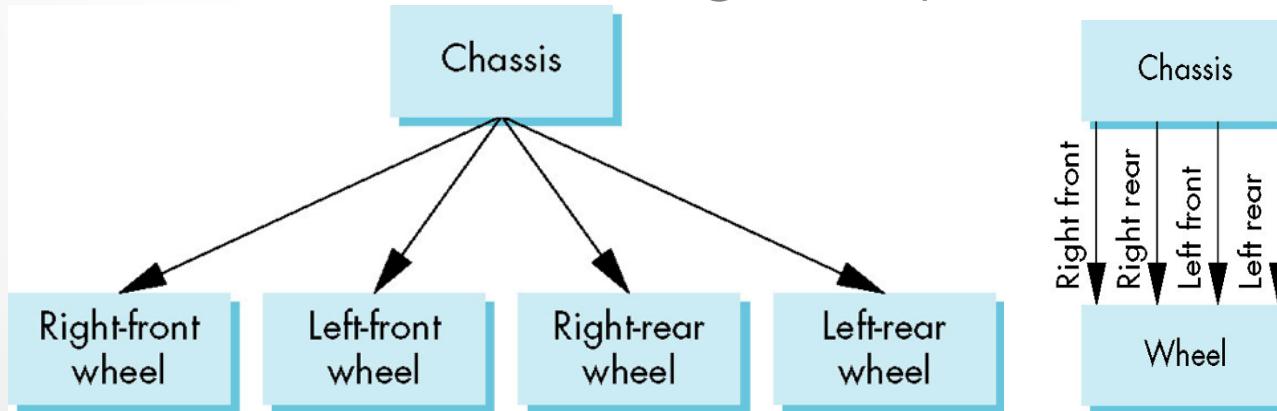
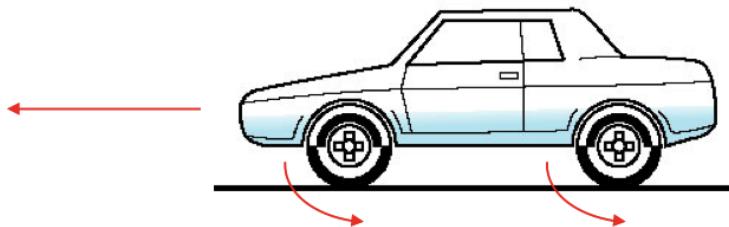
Escena

- Per **estructurar l'escena**: Scene graph:
 - Característiques
 - Eficiència
 - Portabilitat
 - Escalabilitat
 - Nodes: Avatar, Càmera, Llums, Objectes, Transformacions, operacions, scripts, so
 - Defineix relacions: Lògiques, Espaials Funcionals
 - Implementacions (apuntadors)
 - Cerca: és una operació de cost lineal
 - En escenes grans
 - Bounding Volume Hierarchies
 - Space partitioning structures



Escena

- Per **estructurar objectes**: Model símbol-instància
 - Es parteix de símbols bàsics
 - L'objecte és una instanciació de símbols bàsics i les transformacions associades.
 - Exemple: Cotxe
- Visualitzar: recorregut en pre-ordre de l'arbre



Índex

2.1. Introducció

2.2. Elements gràfics

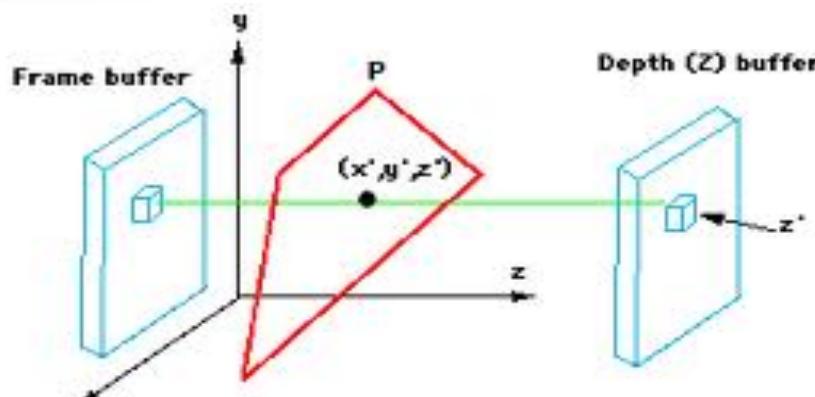
- Objectes
- Càmera
- Llums
- Escena

2.3. Pipeline gràfica

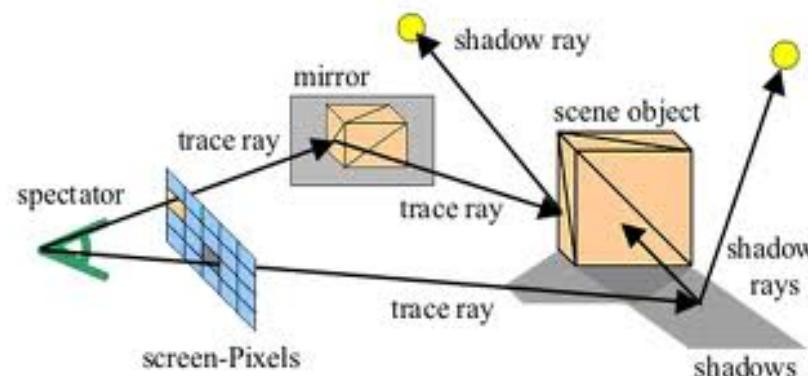
- Arquitectura clàssica
- Arquitectura GPU

Pipeline de visualització

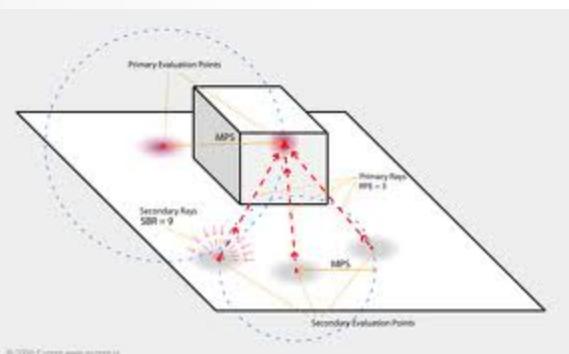
- Projectius



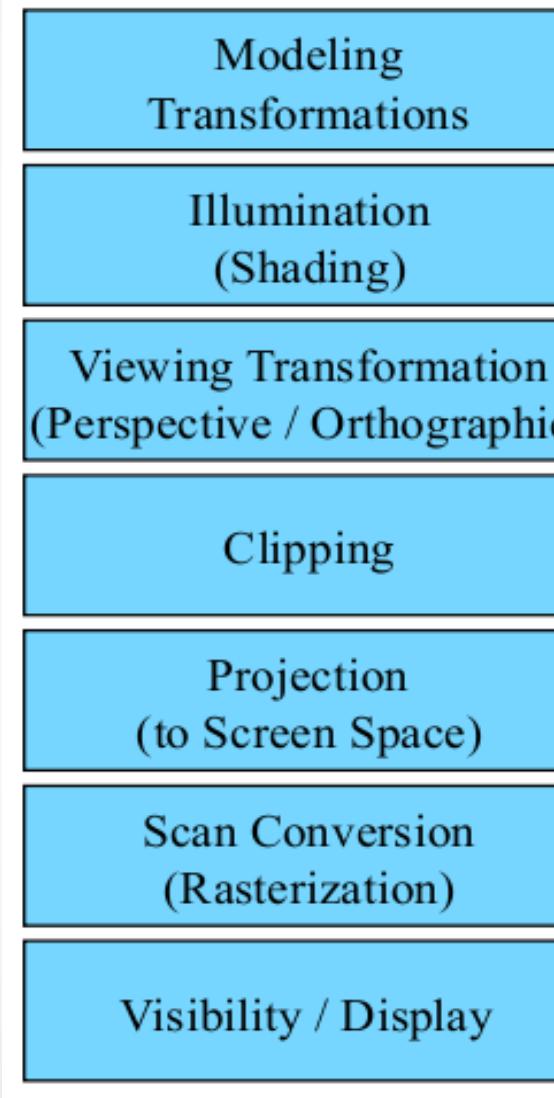
- Raytracing



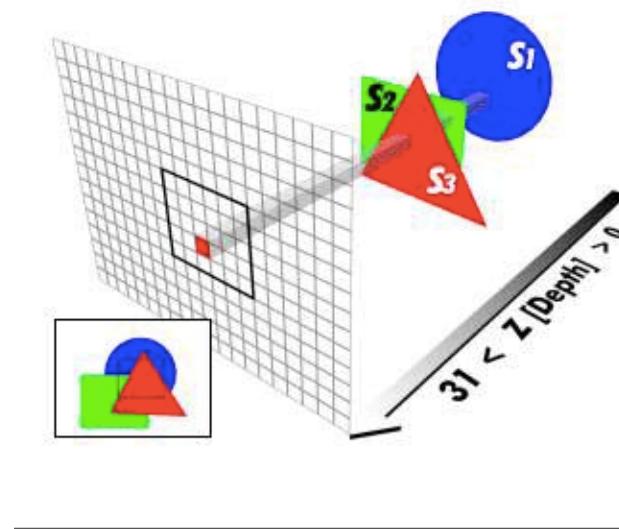
- Radiosity



Pipeline de visualització



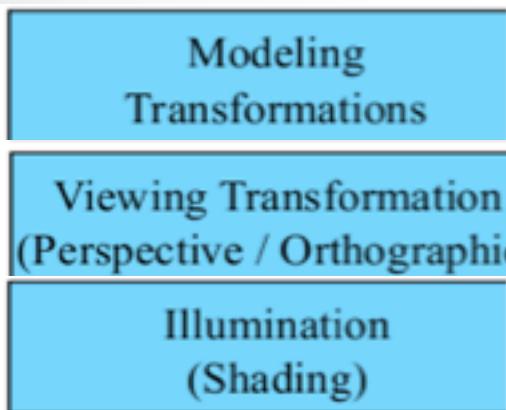
- Mètodes projectius



1	0 0 0 0 0 0 0 0 0 0 0 0
2	0 0 0 0 0 0 0 0 0 0 0 0 10 10 10 10 0 0 10 10 10 10 0 0 10 10 10 10 0 0
3	5 5 5 5 5 5 5 5 5 5 5 5 10 10 10 10 5 5 10 10 10 10 5 5 10 10 10 10 5 5
4	5 5 15 15 5 5 5 5 15 15 15 5 10 15 15 15 15 15 10 15 15 15 15 15 15 15 15 15 15 15

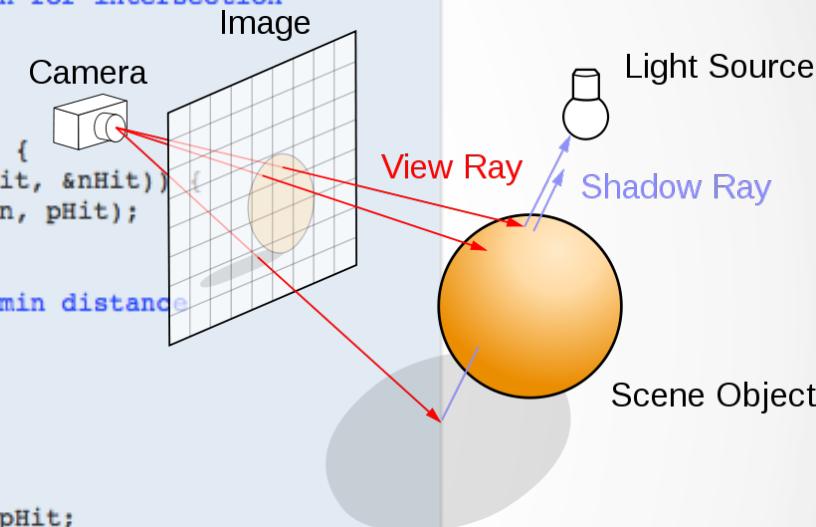
Pipeline de visualització

- Raytracing



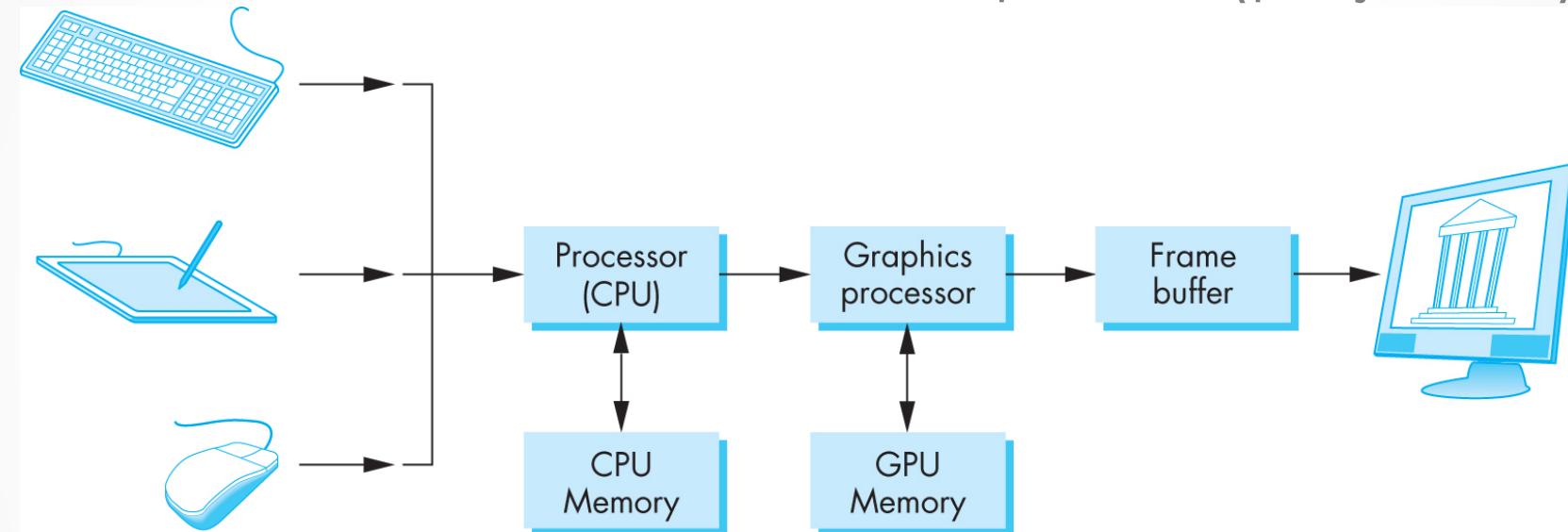
```
1 for (int j = 0; j < imageHeight; ++j) {  
2     for (int i = 0; i < imageWidth; ++i) {  
3         // compute primary ray direction  
4         Ray primRay;  
5         computePrimRay(i, j, &primRay);  
6         // shoot prim ray in the scene and search for intersection  
7         Point pHit;  
8         Normal nHit;  
9         float minDist = INFINITY;  
10        Object object = NULL;  
11        for (int k = 0; k < objects.size(); ++k) {  
12            if (Intersect(objects[k], primRay, &pHit, &nHit)) {  
13                float distance = Distance(eyePosition, pHit);  
14                if (distance < minDistance) {  
15                    object = objects[k];  
16                    minDistance = distance; // update min distance  
17                }  
18            }  
19        }  
20        if (object != NULL) {  
21            // compute illumination  
22            Ray shadowRay;  
23            shadowRay.direction = lightPosition - pHit;  
24            bool isInShadow = false;  
25            for (int k = 0; k < objects.size(); ++k) {  
26                if (Intersect(objects[k], shadowRay)) {  
27                    isInShadow = true;  
28                    break;  
29                }  
30            }  
31        }  
32        if (!isInShadow)  
33            pixels[i][j] = object->color * light.brightness;  
34        else  
35            pixels[i][j] = 0;  
36    }  
37 }
```

<http://www.scratchapixel.com>



Arquitectura clàssica

- Es calculen les transformacions a la CPU i s'envia a la GPU amb crides directes a OpenGL (projectius)



- Hi han maneres d'enviar les dades de cop a la GPU, via *display lists* o *vertex arrays*
- La rasterització i el display es fa a la GPU

Arquitectura clàssica

Exemple d'arquitectura clàssica OpenGL:

```
glClear( GL_COLOR_BUFFER_BIT );           /* color de background */
glEnable(GL_DEPTH_TEST);                 /* activació del test d'eliminació de parts amagades */
glShadeModel(GL_FLAT);                  /* model d'il.uminacio */

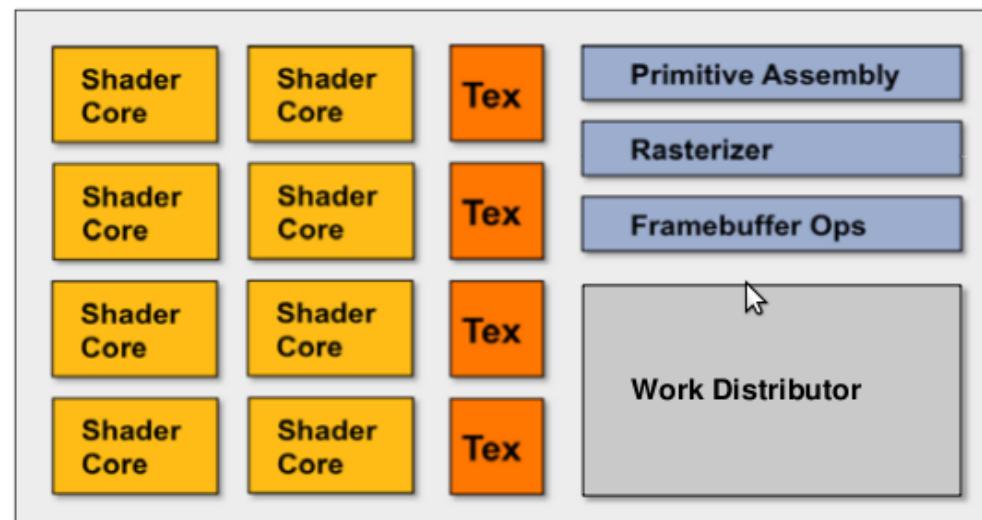
glMatrixMode( GL_PROJECTION );          /* Matriu de projeccio */
glLoadIdentity();                        /* Inicialitzacio de la matriu de projeccio */
                                        /* identity */
glFrustum( -1, 1, -1, 1, 1, 1000 );    /* Projeccio perspectiva */

glMatrixMode( GL_MODELVIEW );          /* Model view */
glLoadIdentity();                        /* Inicialitzacio de la matriu de visio */
glTranslatef( 0, 0, -3 );                /* Traslacio dels vertexs, encara que també es
                                            poden rotar segons la posicio de la camera */

glBegin( GL_POLYGON );                /* Pintat de vertexs */
glColor3f( 0, 1, 0 );                  /* Set the current color to green */
glVertex3f( -1, -1, 0 );               /* Issue a vertex */
glVertex3f( -1, 1, 0 );                /* Issue a vertex */
glVertex3f( 1, 1, 0 );                 /* Issue a vertex */
glVertex3f( 1, -1, 0 );                /* Issue a vertex */
glEnd();                                /* Enviament de tot a la targeta gràfica */
```

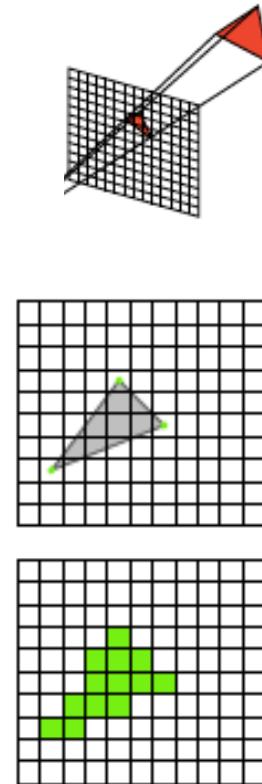
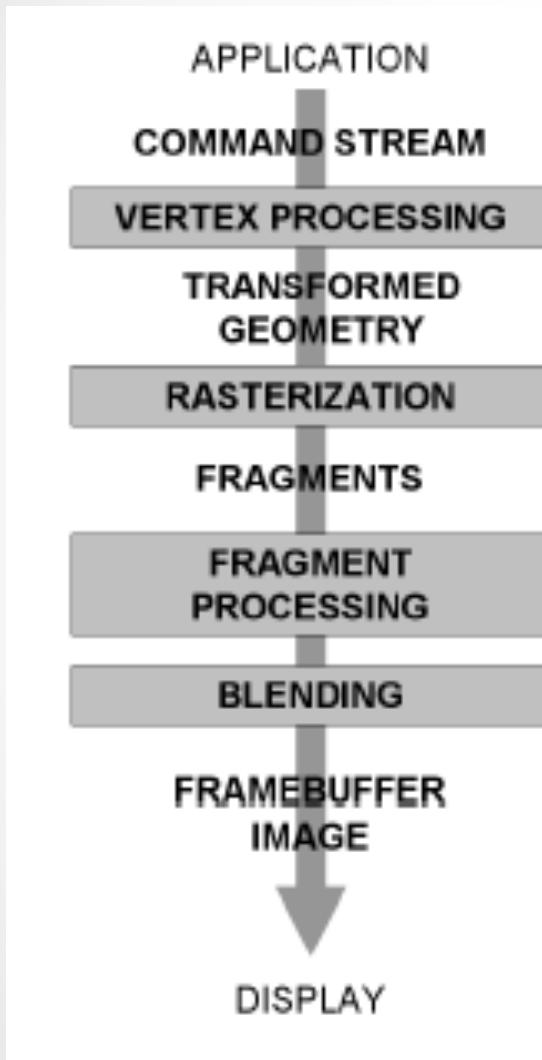
Arquitectura GPU

- Com està dissenyada la GPU?
 - La GPU està formada per un conjunt de processadors, que permeten paral·lelitzar a diferents nivells (són els shader cores)
 - Es tracten en paral·lel els vèrtexs que s'envien
 - Es paral·lelitzen els fragments que es generen internament



Per una explicació més detallada consulteu la secció 1.7 del llibre [Angel2011]

Arquitectura GPU

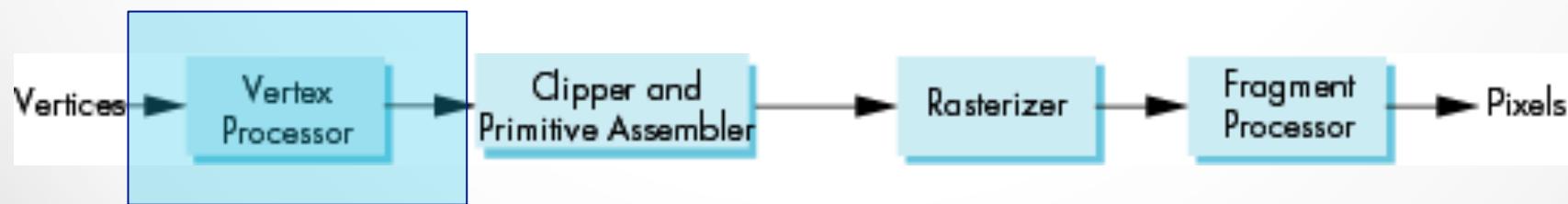


- Aplicació:
- **Vertex processing:**
- Transformació de la geometria 3D a 2D +clipping
- Rasterization:
 - De la geometria 2D a fragments
- **Fragment processing i blending:**
 - Processat de segments i combinacions a nivells de píxels



Arquitectura GPU

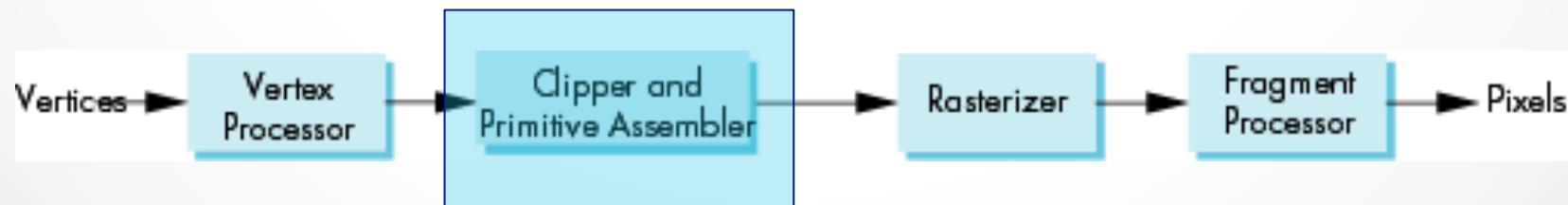
- **Vertex Processor:** Programa que s'aplica a cada vèrtex. S'utilitza per realitzar el canvi de sistemes de coordenades:
 - Coordenades de món
 - Coordenades de càmera
 - Coordenades de pantalla
- Cada canvi de coordenades equival a una transformació amb matrius
- En el vertex processor també es calcula el color



Arquitectura GPU

Clipping i Primitive Assembler: En aquesta etapa es realitzen les següents etapes:

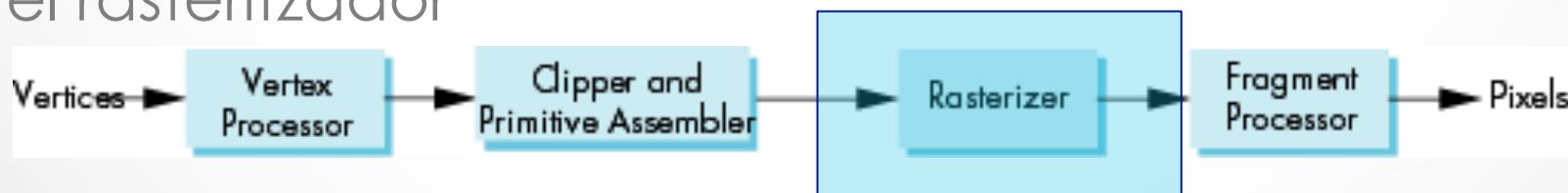
- Els vèrtexs transformats es combinen en primitives bàsiques:
 - 1 vèrtex -> punt
 - 2 vèrtexs -> línia
 - 3 vèrtexs -> triangle
- Es fa el retallat (*clipping*)
- Es realitzen les projeccions (axonomètrica o perspectiva)
- Es transforma a coordenades de viewport



Arquitectura GPU

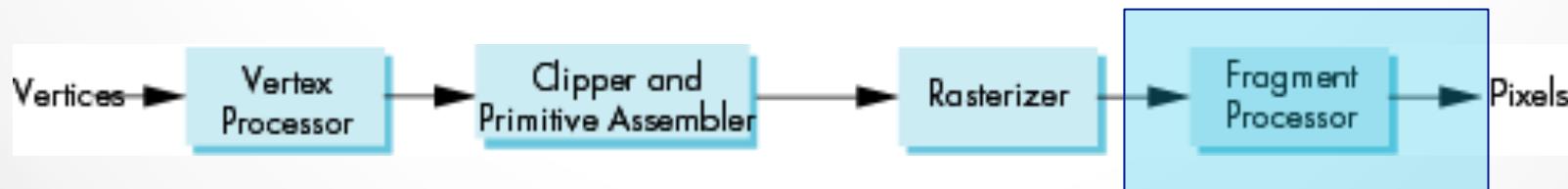
Rasterization: Per a cada triangle generat en les fases anteriors:

- Es mostreja el triangle i es divideix en fragments de píxels
- S'interpolen colors i coordenades
- Els fragments són “píxels potencials”:
 - Tenen una posició en el *frame buffer*
 - Tenen color i profunditat (*depth*)
- Els atributs dels vèrtexs s'interpolen sobre els objectes en el rasteritzador



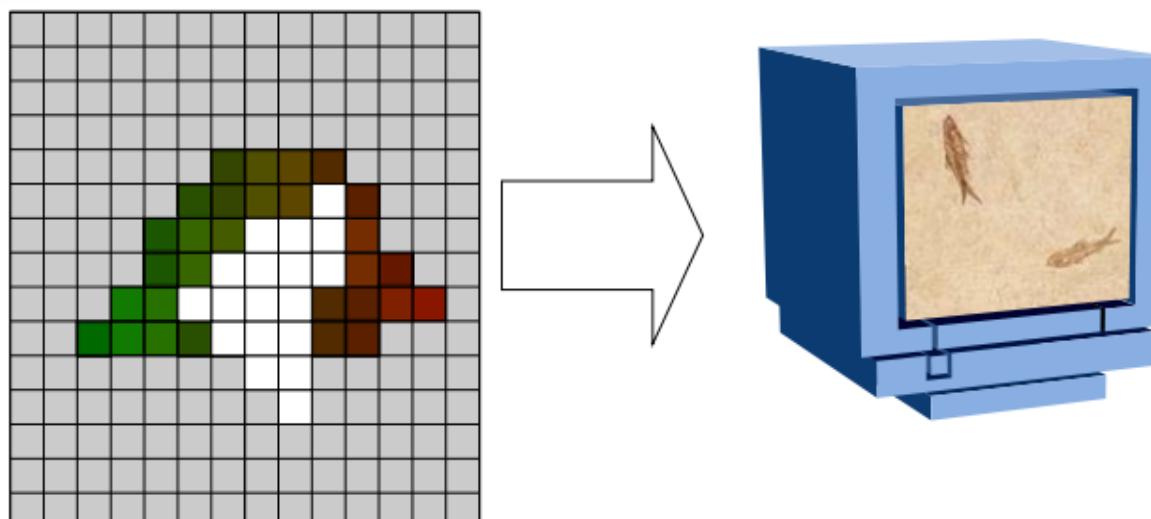
Arquitectura GPU

- Cada fragment generat es processa en paral·lel en el **Fragment Processor**.
- Cada fragment es processa per determinar el color en el píxel corresponent al *frame buffer*
- Els colors es poden determinar per textures o per interpolació dels colors dels vèrtexs
- Els fragments poden ser eliminats per altres fragments més propers a la càmera (eliminació de parts amagades)



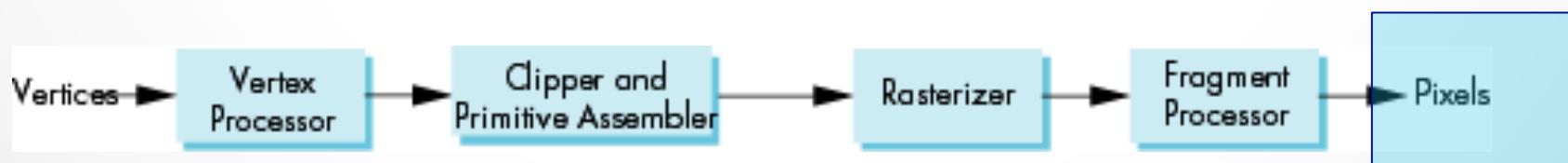
Arquitectura GPU

- Finalment, se li aplica una **correcció gamma** al frame buffer generat i s'envia al display físic



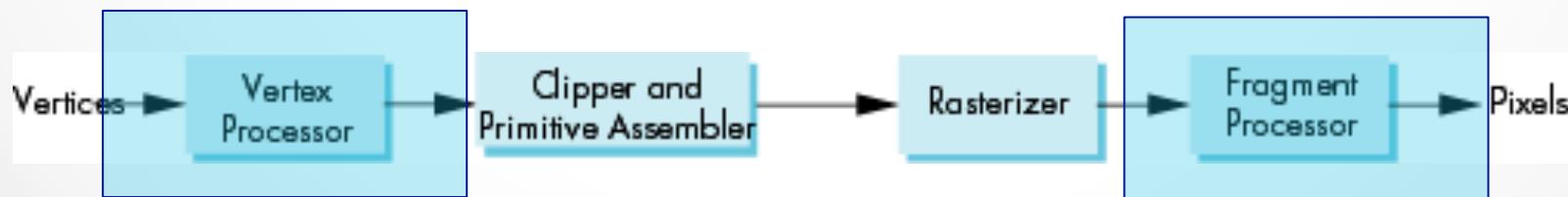
Framebuffer Pixels

Light



Arquitectura GPU

- Les etapes programables són:
 - Vertex processor
 - Fragment processor
- Es programen amb un llenguatge anomenat **GLSL**, molt semblant al C
- Els programes s'executen en els *Shader cores* de la GPU
- Aquests programes s'anomenen *vertex shader* i *fragment shader*, respectivament



Per una explicació més detallada consulteu la secció 2.8 del llibre [Angel2011]

Arquitectura GPU

Inicialització dels shaders des del programa de la **CPU**:

- Es carreguen, compilen i linken en temps d'execució el programa escrit en C, C++, phyton, etc.

```
void GLWidget::InitShader(const char* vShaderFile,  
                           const char* fShaderFile)
```

- S'invoquen de forma automàtica cada vegada que es fa una crida de pintat de GL (`glDraw*`)
- S'executen amb els valors que s'han connectat des del programa (`toGPU`)

Arquitectura GPU

- Connexió de variables des de la **CPU** als shaders

// Creació i inicialització d'un buffer de GL en memòria dels shaders

```
glGenBuffers( 1, &buffer );
 glBindBuffer( GL_ARRAY_BUFFER, buffer );
 glBufferData( GL_ARRAY_BUFFER, sizeof(points) + sizeof(colors), NULL, GL_STATIC_DRAW );
 glBufferSubData( GL_ARRAY_BUFFER, 0, sizeof(points), points );
 glBufferSubData( GL_ARRAY_BUFFER, sizeof(points), sizeof(colors), colors );
```

// connexió amb el vertex/fragment shader

```
int vertexLocation = program->attributeLocation("vPosition");
program->enableAttributeArray(vertexLocation);
program->setAttributeBuffer("vPosition", GL_FLOAT, 0, 4);

int colorLocation = program->attributeLocation("vColor");
program->enableAttributeArray(colorLocation);
program->setAttributeBuffer("vColor", GL_FLOAT, sizeof(points), 4);
program->bindAttributeLocation("vPosition", vertexLocation);
program->bindAttributeLocation("vColor", colorLocation);
```

// linkatge i bind del programa al codi C++

```
program->link(); program->bind();
```

Arquitectura GPU

- Exemple de vertex shader: (vshader1.glsl)

```
#version 150
in vec4 vPosition;
in vec4 vColor;
out vec4 color;
void main()
{
    gl_Position = vPosition;
    color = vColor;
}
```

- Exemple de fragment shader: (fshader1.glsl)

```
in vec4 color;
out vec4 colorOut;
void main()
{
    colorOut = color
}
```

Conclusions

- En aquest tema has après els **elements** i les etapes implicades en el **pipeline** (o procés) de **visualització** de qualsevol aplicació gràfica.
- Has explorat com el pipeline de visualització s'implementa en l'arquitectura centrada en GPUs
- Podrás entendre el codi d'un shader bàsic. A partir d'aquí analitza els shaders que s'han facilitat a la pràctica 0.