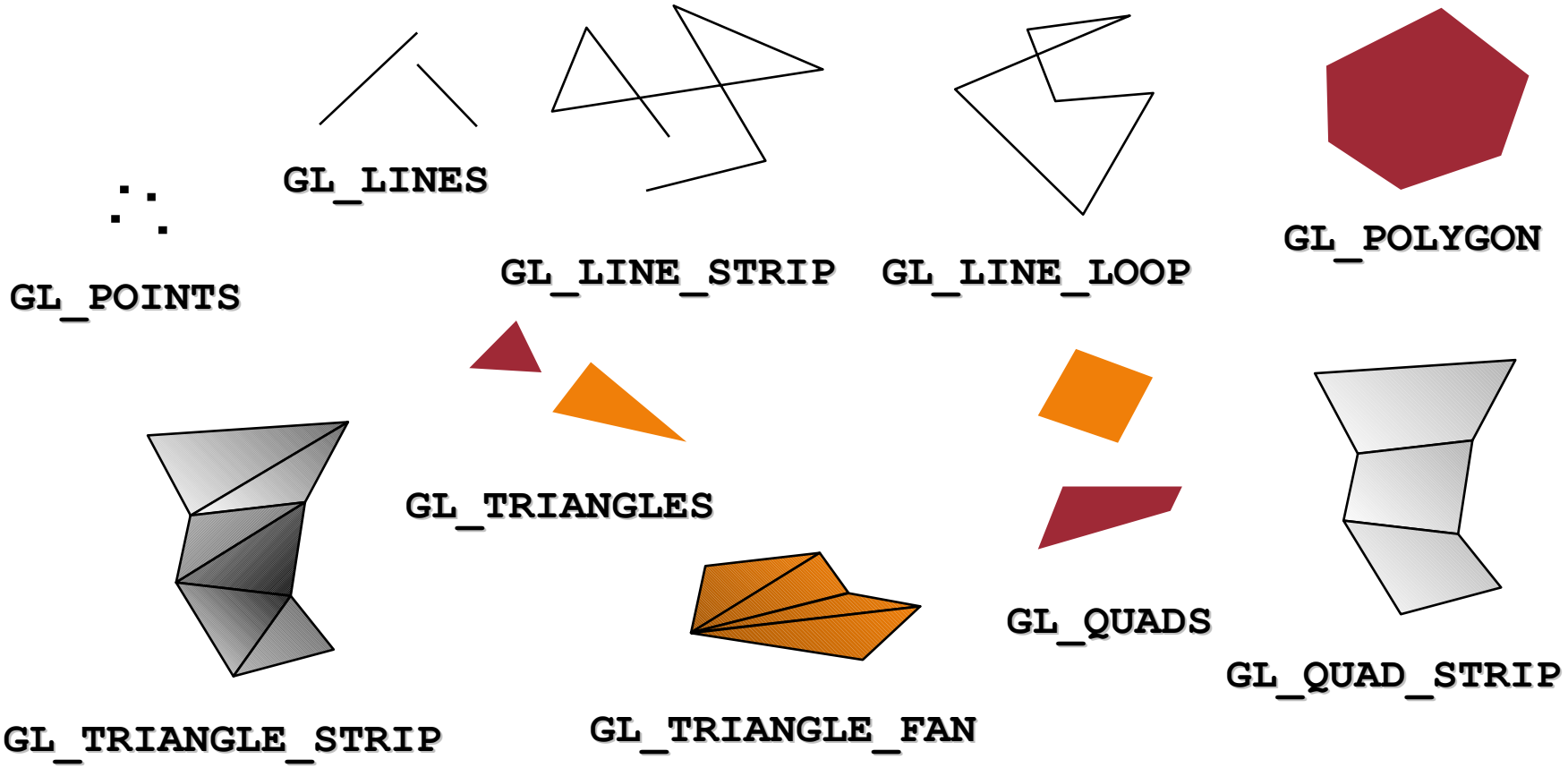
Introducción a la programación en OpenGL (2)

Primitivas

- Todas las primitivas son definidas por sus vértics



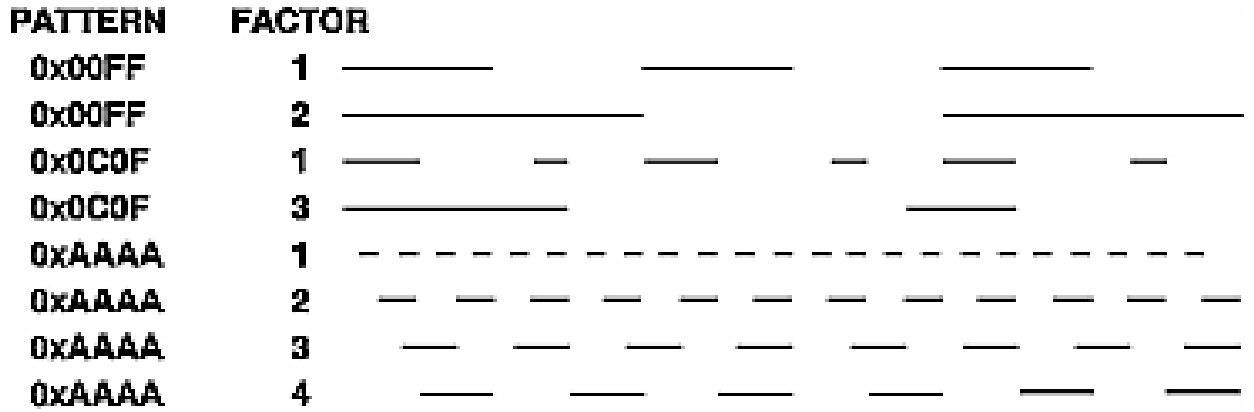
Características a definir

- **Puntos:** se puede especificar el tamaño de los puntos dibujados con *GL_POINTS*
  - `glPointSize(GLfloat size_in_pixels);`

- Usar un tamaño diferente a 1 tiene diferentes efectos dependiendo si se ha habilitado el antialiasing de puntos.
  - `glEnalbe(GL_POINT_SMOOTH);`

Características a definir

- **Líneas:** se puede especificar el grosor de las líneas dibujadas con *GL_LINES*
  - `glLineWidth(GLfloat with_in_pixels);`
- También se puede especificar un patrón:
  - `glLineStipple(GLint factor,GLushort pattern);`
  - `glEnable(GL_LINE_STIPPLE);`

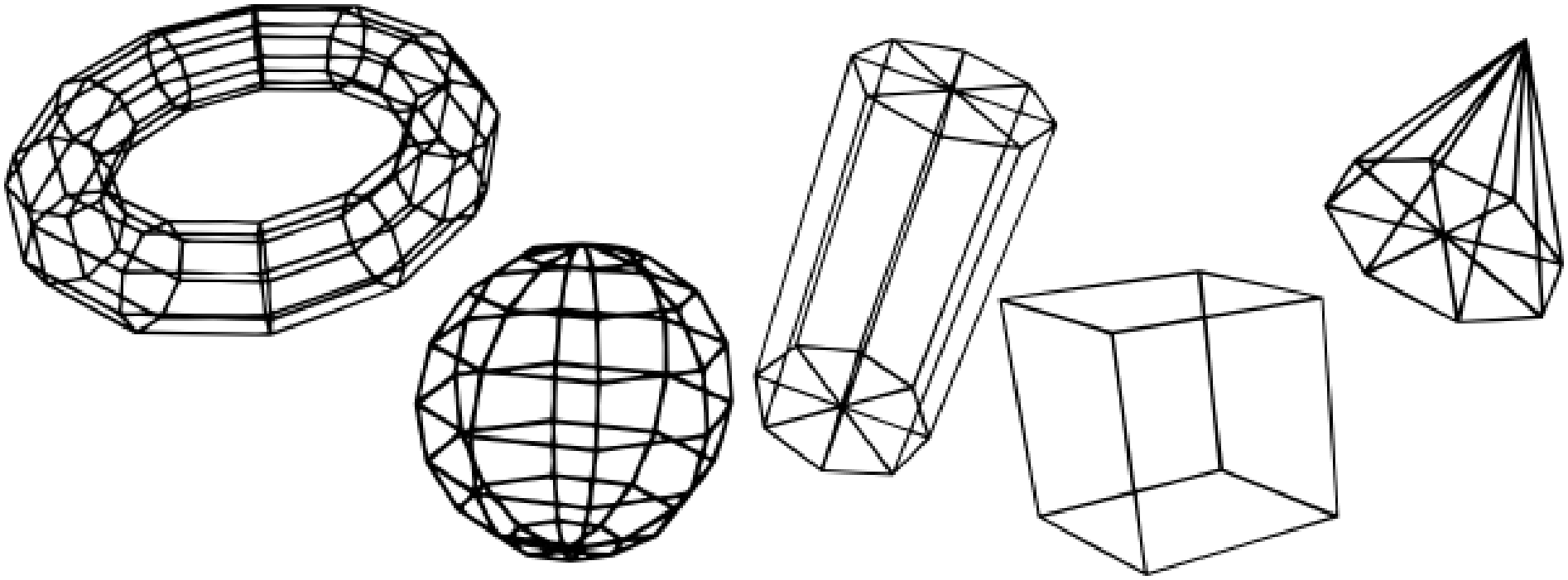


Características a definir

- **Polígonos:** se puede especificar la forma en la que serán dibujados los polígonos
  - `glPolygonMode(GLenum face,GLenum mode)`
  - **face:** se refiere a la cara del polígono
    - `GL_FRONT_AND_BACK`
    - `GL_FRONT`
    - `GL_BACK`
  - **mode:** se refiere a la forma de dibujado
    - `GL_POINT`
    - `GL_LINE`
    - `GL_FILL`

Características a definir

- **Wireframe:** es un modo de dibujado con un nombre propio. Sólo se dibujan las aristas:
  - `GlPolygonMode( GL_FRONT_AND_BACK, GL_LINES);`



Características a definir

- Se puede especificar como OpenGL determina la cara frontal de un polígono

- `glFrontFace(GLenum mode);`
  - `GL_CCW`: antihorario (por defecto)
  - `GL_CW`: horario

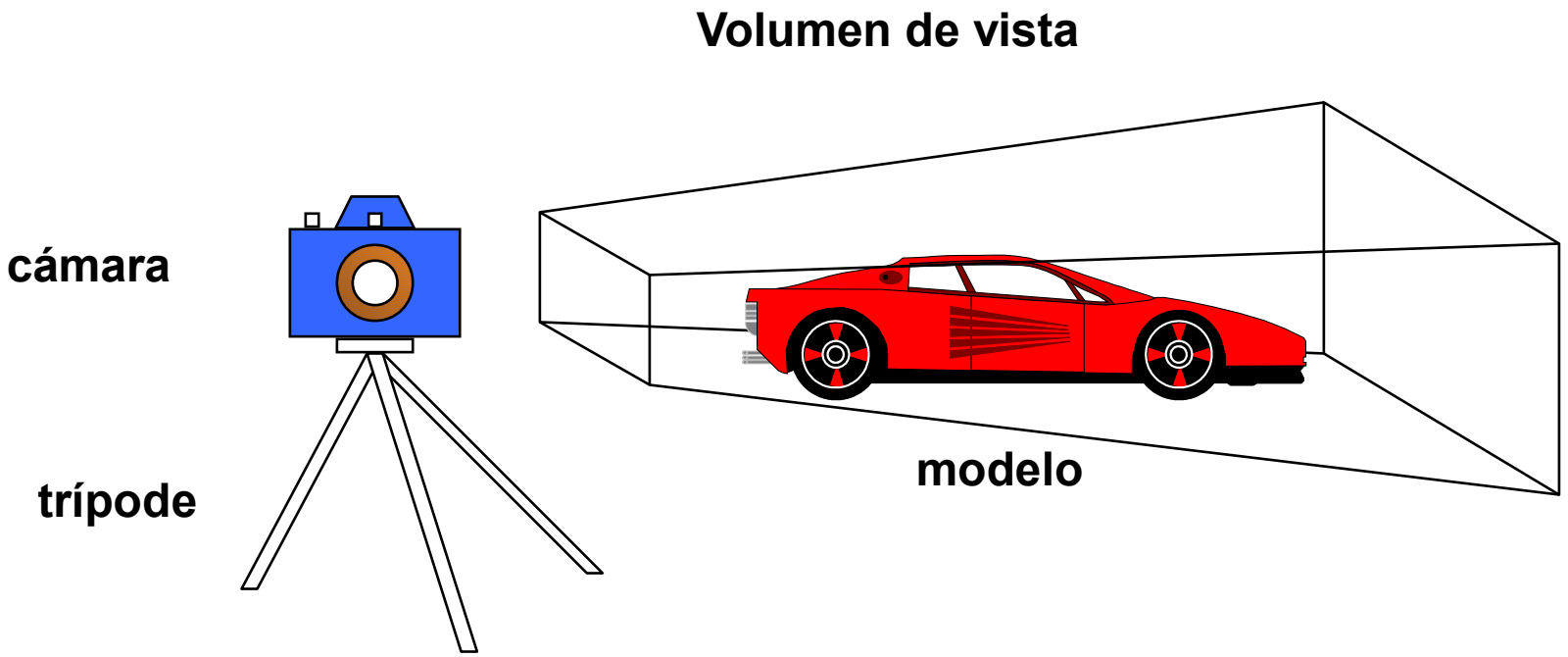
Backface culling

- Es una técnica que permite acelerar el dibujado de polígonos eliminando aquellos cuya cara frontal NO sea vista. Ésto pasa cuando se usan polígonos para definir superficies cerradas.
  - Todo polígono cuya cara frontal “no sea visible”, se encontrará oculto por otros polígonos cuya cara frontal “SÍ son visibles”
- `glCullMode(GLenum mode);`
  - `GL_FRONT`
  - `GL_BACK` (valor por defecto)
  - `GL_FRONT_AND_BACK`
- `glEnable(GL_CULL_FACE);`

Transformaciones Geométricas

- Se usan para:
  - definir la vista
    - orientación de la cámara
    - definir el tipo de proyección
  - posicionar los objetos en la escena
  - mapeo a la pantalla

Analogía con la Cámara

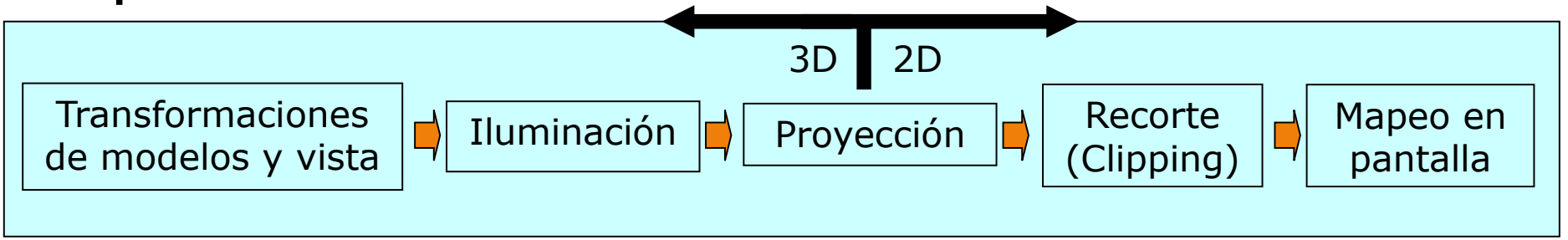


Analogía con la Cámara

- Transformaciones de proyección
  - Ajustar los lentes de la cámara
- Transformaciones de vista
  - Se define la posición y orientación del volumen de vista en el mundo: posicionar la cámara
- Transformaciones del modelo
  - Posicionar el modelo en el mundo
- Transformaciones del Viewport
  - Agrandar o reducir la fotografía física

Transformaciones Geométricas

- Pipeline funcional:



- **Objetivo**:
  - Realizar transformaciones sobre los objetos de modo de llevar su descripción tridimensional a una bidimensional evaluando su iluminación y eliminando los que no se vean.
  - **Nota:** la iluminación se calcula antes de que se realice el clipping o se utilice el z-buffer para determinar si el polígono es visible!!

Transformaciones Geométricas

- Modelos:
  - Originalmente los objetos están definidos en su sistema local de coordenadas (model space)
  - Se les aplica una transformación que los orienta y posiciona (model tranform) definida por una matriz 4x4 homogénea. Se transforman sus vértices y normales.
  - Se obtienen coordenadas del modelo en el espacio global absoluto (world space)

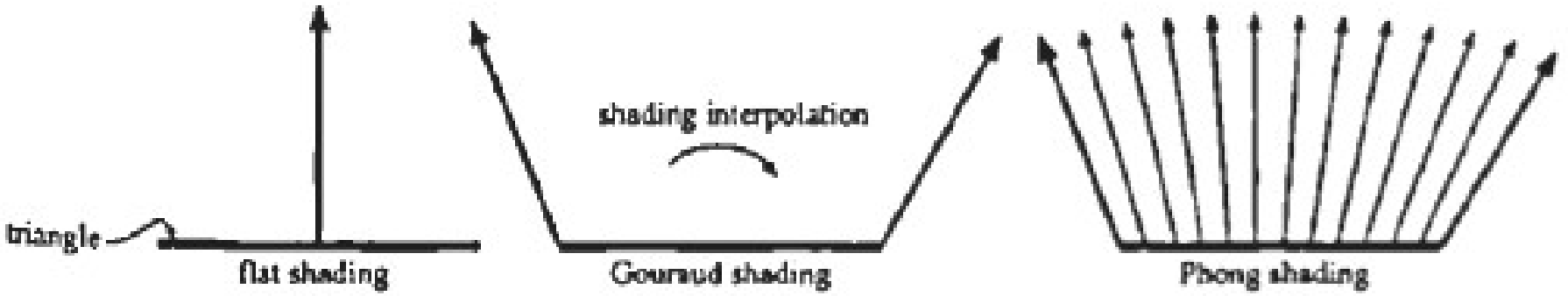
Modelos de color

- Se puede trabajar con dos modos de color:
  - `glShadeModel( GLenum mode );`
  - **mode**:
    - `GL_FLAT`: asigna un único color a toda la superficie del polígono
    - `GL_SMOOTH`: el color interior al polígono es el resultado de interpolar el color de sus vértices (Gouraud Shading)

Iluminación

- Estado que se encarga de los cálculos a realizarse cuando hay definidas luces en el sistema.
- Se aplica a **TODOS** los vértices transformados, **también a los no visibles!**
- Utiliza el modelo de iluminación clásico implementado por las APIs: ecuación de la luz
- Se pueden utilizar otros modelos de iluminación
  - Utilizando Pixel y Vertex Shaders

Iluminación



Sólo con técnicas avanzadas

(Pixel Shader y Vertex Shader)

Mapeo de Texturas

- Sistema mediante el cual se puede aplicar una imagen a un polígono
- Permite aumentar el realismo de la geometría

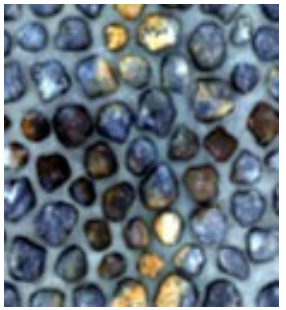
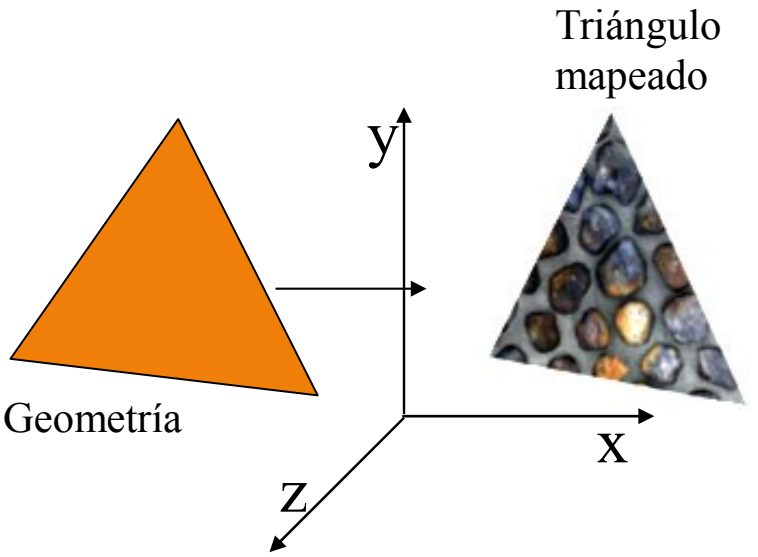
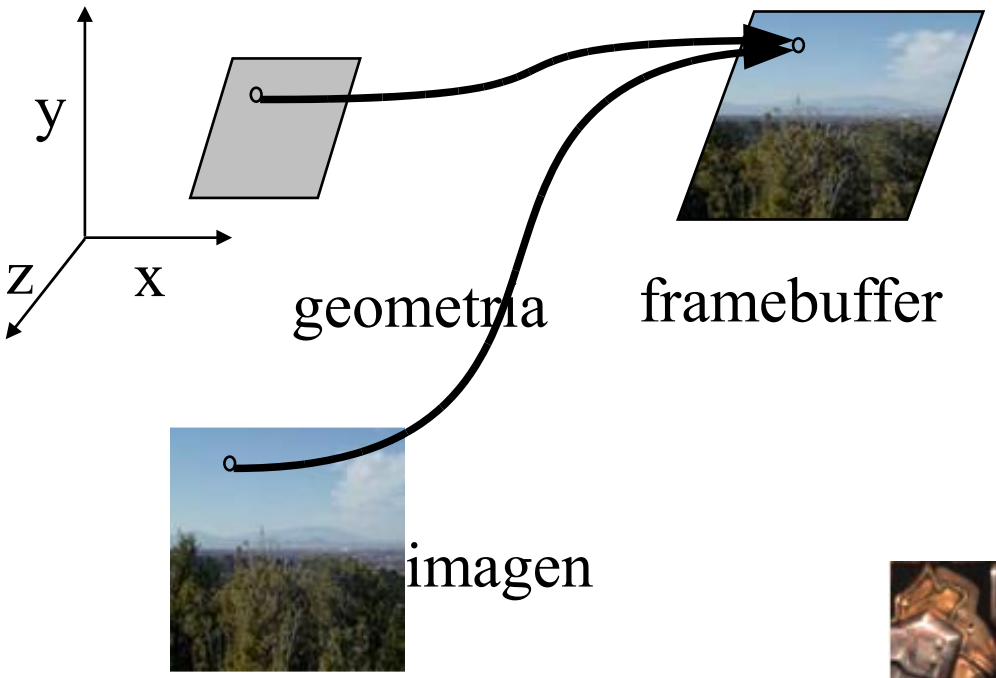


Imagen original: archivo *.jpg, *.png, etc.



Mapeo de Texturas

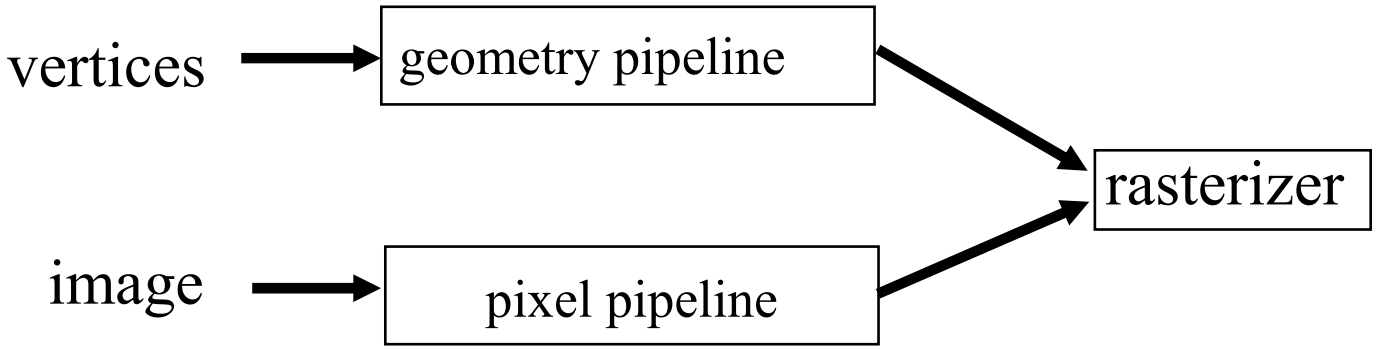


Sarge (Q3A)

Textura (cuerpo)

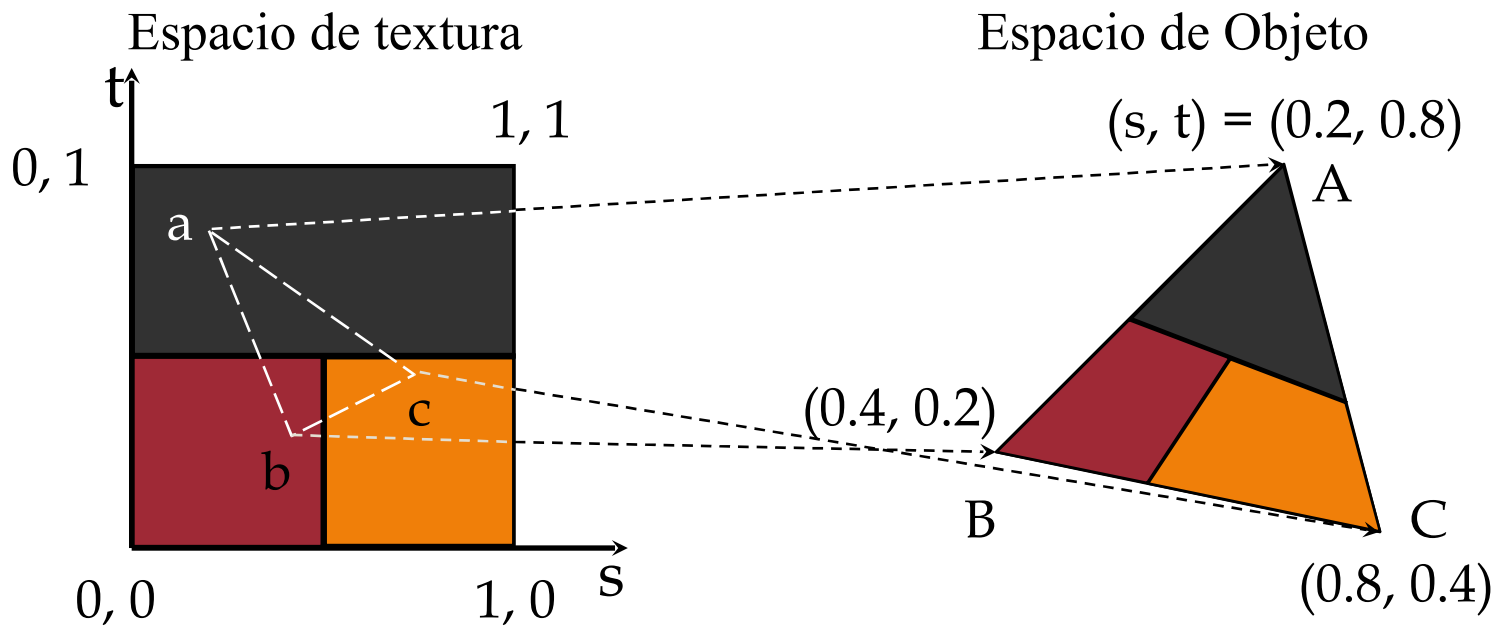
Mapeo de Texturas

- Imágenes y geometría fluyen desde ductos separados que se juntan en el rasterizador
- Esto significa que puede tratarse al mapeo de texturas como un estado más de OpenGL



Mapeo de Texturas

- El mapeo de texturas se especifica por cada vértice.
- Las coordenadas de texturas están en el rango [0, 1]

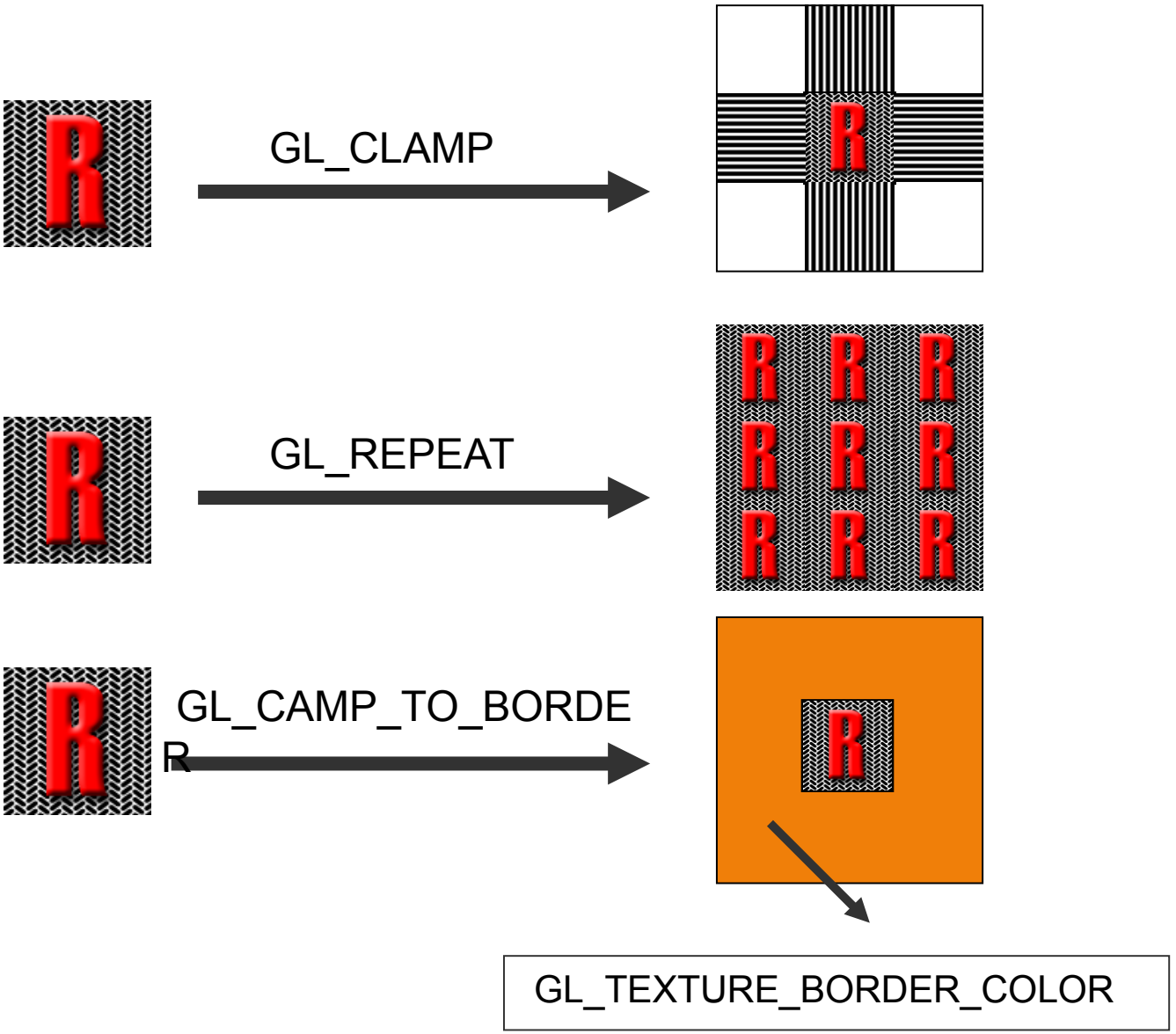


Envolventes

- Cuando las coordenadas de textura se encuentran fuera del intervalo [0, 1] es necesario indicar como se tiene que dibujar el polígono
  - `glTexParameteri(GL_TEXTURE2D,type,mode)`

type	mode
GL_TEXTURE_WRAP_S	GL_CLAMP, GL_REPEAT, GL_CLAMP_TO_BORDER_ARB, GL_CLAMP_TO_EDGE
GL_TEXTURE_WRAP_T	GL_CLAMP,GL_REPEAT, GL_CLAMP_TO_BORDER_ARB, GL_CLAMP_TO_EDGE
GL_TEXTURE_BORDER_COLOR	RGBA en el rango [0,1]

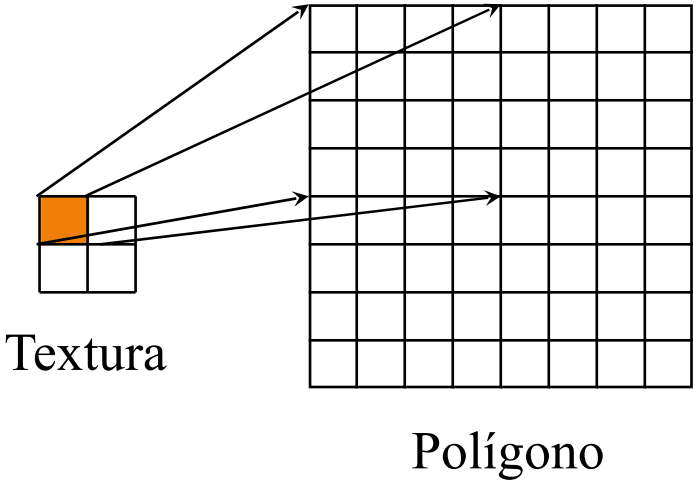
Envolventes



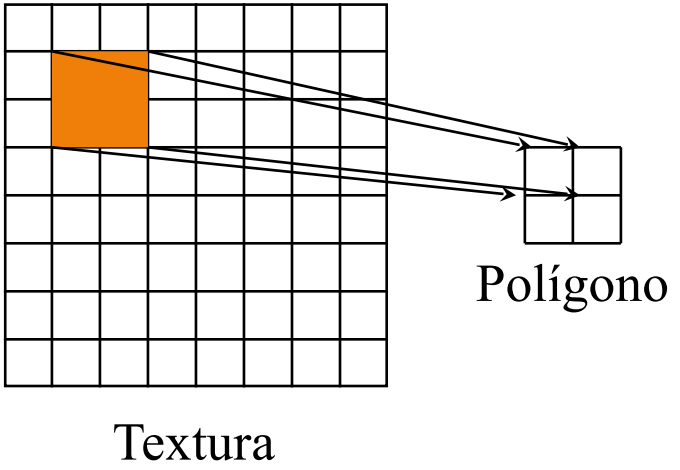
Envolventes

Filtrado

- Generalmente las texturas y los polígonos no tienen las mismas dimensiones
  - La correspondencia de texels y pixels no es 1 a 1
- OpenGL debe aumentar o disminuir los texels para aplicarlos a la geometría



Magnificación: un pixel de la textura (texel) genera varios en el polígono final



Minimización: muchos texels de la Textura caen en el mismo píxel del polígono

Filtrado

- La definición del filtrado de texturas se hace con la función
  - `glTexParameteri(GL_TEXTURE_2D,type,mode);`
  - type:
    - `GL_TEXTURE_MIN_FILTER`
    - `GL_TEXTURE_MAX_FILTER`
  - mode:
    - `GL_NEAREST`: no interpola, elije el texel más cercano
    - `GL_LINEAR`: interpola los colores de los pixels del polígono usando la información de los colores de los texels adyacentes

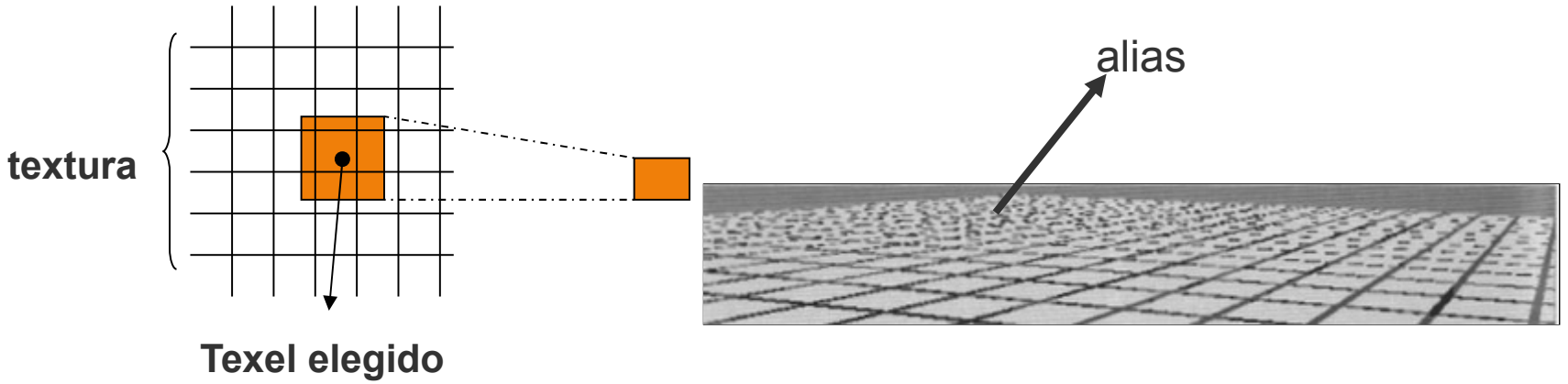
Filtrado

GL_NEAREST

GL_LINEAR

MipMapping

- En la magnificación la interpolación lineal funciona bastante bien.
- Sin embargo en la minimización no ocurre lo mismo: la interpolación genera “alias” cuando la geometría es lejana (hay que filtrar más)

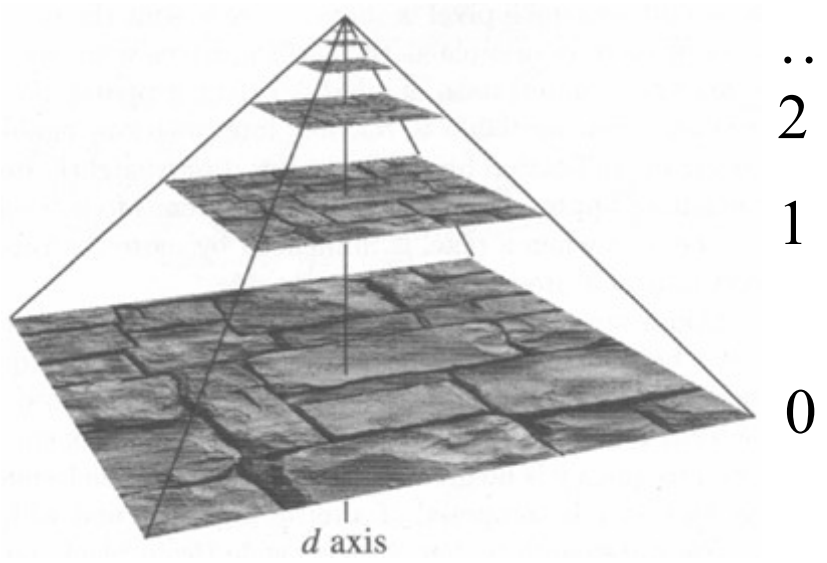


MipMapping

- Un filtrado de más calidad que funcionara bien para todos los grados de escala implica un pocesamiento de un nivel que es muy costoso de realizar en tiempo real.

- **Solución**: realizar el cálculo off-line utilizando texturas filtradas con diferente grado de minimización.

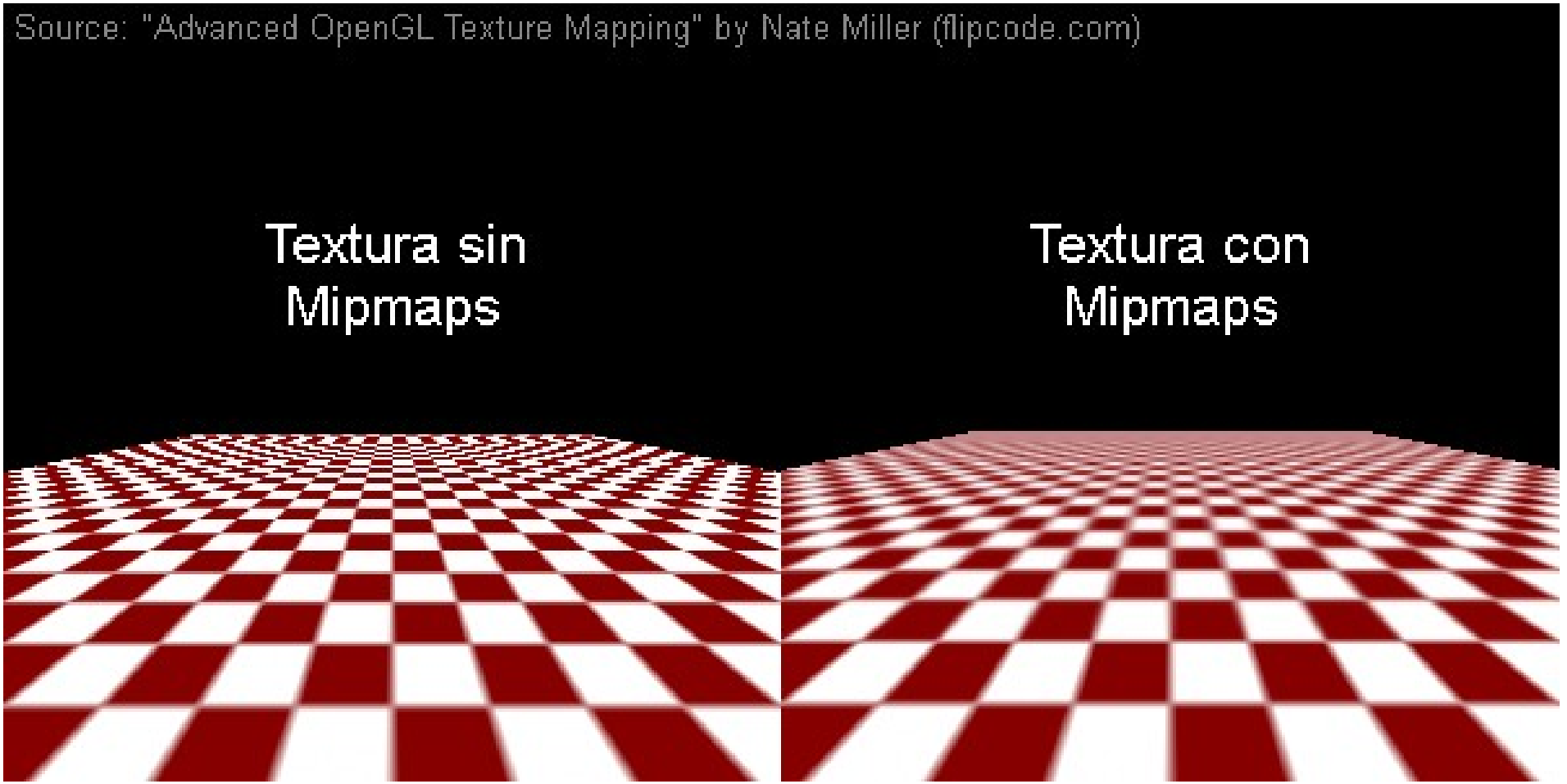
MipMapping



La textura original (level 0) es utilizada para generar diferentes niveles a ¼, 1/8, 1/16, etc.

- En el momento de aplicar las texturas OpenGL decide el nivel a utilizar en función de la cantidad de minimización a aplicar.
- Este cálculo es soportado directamente por hardware.

MipMapping


Source: "Advanced OpenGL Texture Mapping" by Nate Miller (flipcode.com)
Textura sin Mipmaps
Textura con Mipmaps

MipMapping

- Glu provee una primitiva que genera los mipmaps directamente a partir de una textura base y **carga los datos en la memoria de video:**
  - `gluBuild2DMipmaps(GL_TECTURE_2D, 4 ancho, alto,..., datos);`
  - La imagen corresponde al nivel 0
- Si no se quiere usar Mipmaps la primitiva a usar es
  - `glTexImage2D(GL_TEXTURE_2D, 4, ancho, alto,..., datos);`

Objetos de Textura

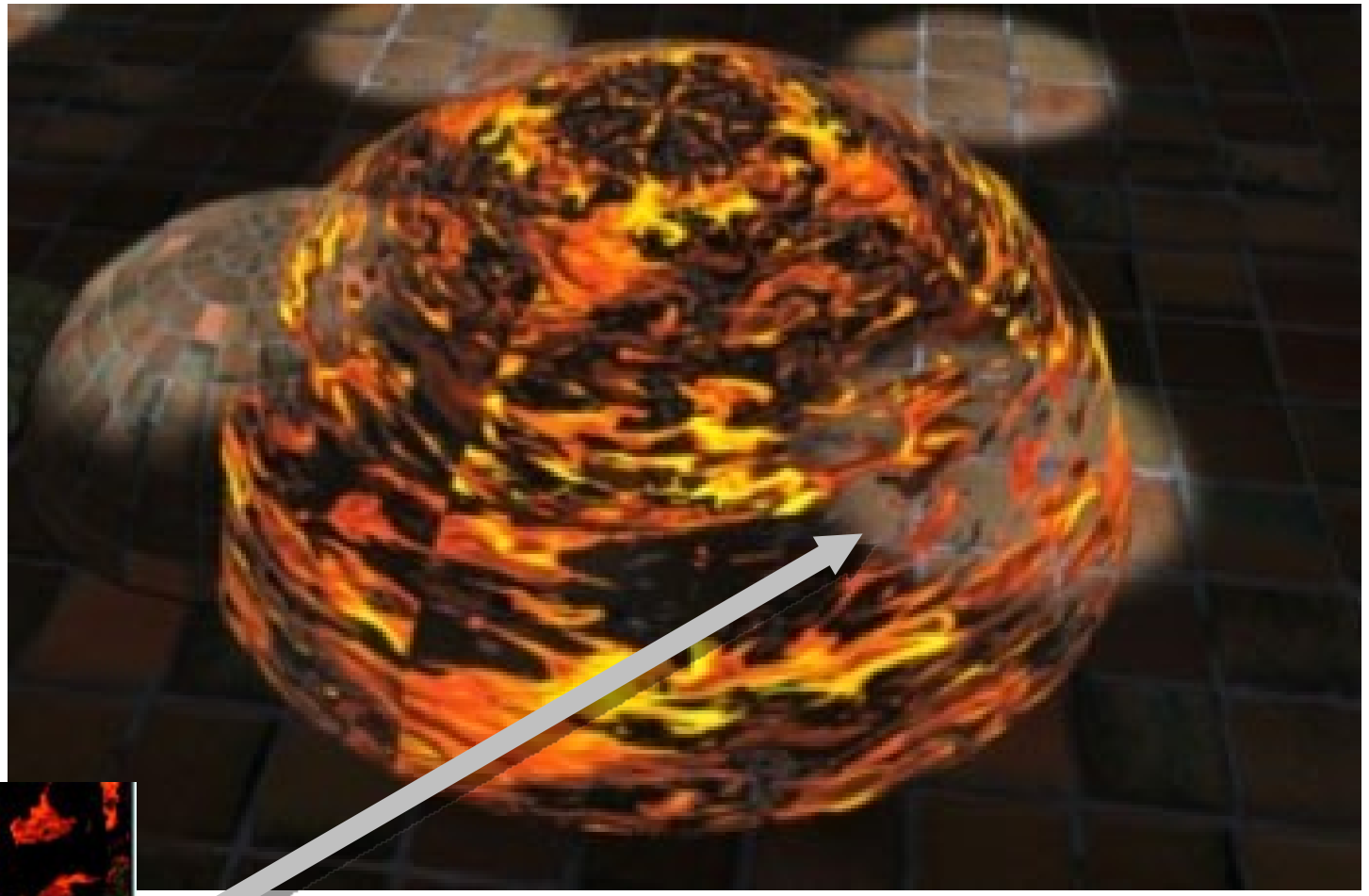
- A cada textura se le asigna un identificador (int) que se utiliza para indicarle al OpenGL cual es la textura con la que estamos trabajando.
- Para la generación de identificadores
  - `glGenTextures( n, *textIds );`
  - Se piden **n** identificadores consecutivos, y se almacena el primer identificador en **textIds**
- Cada vez que queremos definir una textura o utilizarla
  - `glBindTexture( GL_TEXTURE_2D, id );`

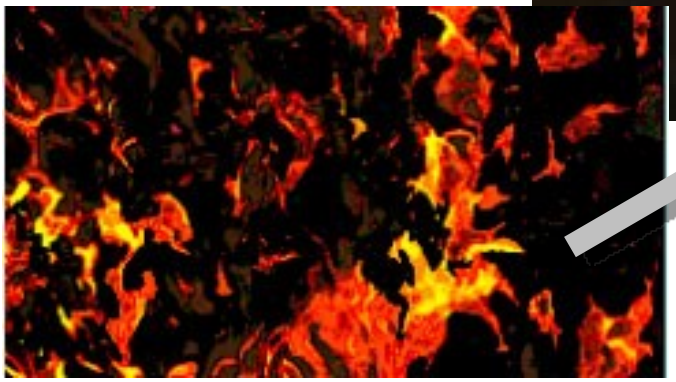
Funciones de combinación

- Cuando se le aplica una textura a un polígono, el mismo ya tiene un color y hay que indicarle a OpenGL como combinarlo con el color de la textura.
  - `GlTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MOVE,param);`
  - **param**:
    - `GL_REPLACE`: la textura reemplaza los otros colores.
    - `GL_DECAL`: igual que `GL_REPLACE` pero considerando valores alpha de las texturas.
    - `GL_MODULATE`: se multiplica el color del polígono por el de la textura utilizando la información de iluminación.

Funciones de combinación

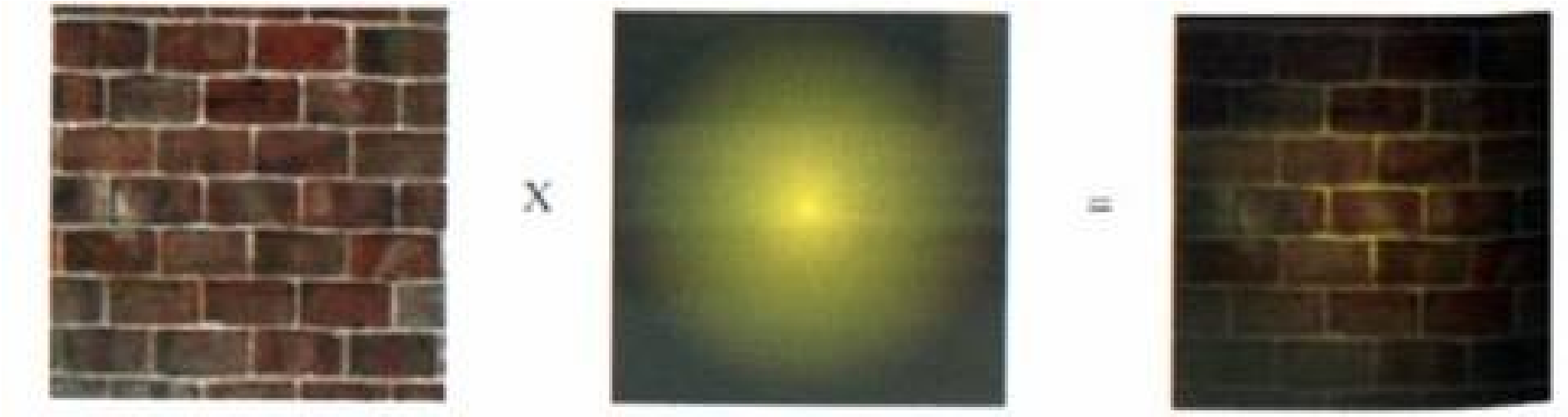
GL_DECAL





Funciones de combinación

GL_MODULATE



- **Nota:** en este ejemplo el efecto de spot se logra teselando la geometría y aplicando un sombreado Gourdaud

Transparencias (blend)

- OpenGL permite controlar la forma en la que los pixels de las primitivas que estamos dibujando son mezclados con los ya existentes en el *colorbuffer*.

- Para ello es necesario activar la funcionalidad de *blending* usando:
  - `glEnable(GL_BLEND);`

Factores de combinación

- Durante el *blending* los pixels de la primitiva (fuente) son combinados con los ya existentes (destino) en el buffer.
- La forma como se combinan se define usando:
  - `glBlendFunc( factorFuente, factorDestino);`
- Cada factor es un valor RGBA que multiplica a los valores fuente y destino respectivamente.
- El valor final se obtiene sumando ambos resultados:
  - final=(fuente*factorFuente + destino*factorDestino)

Constantes de combinación

- La siguiente tabla muestra los diferentes factores de combinación que utiliza OpenGL

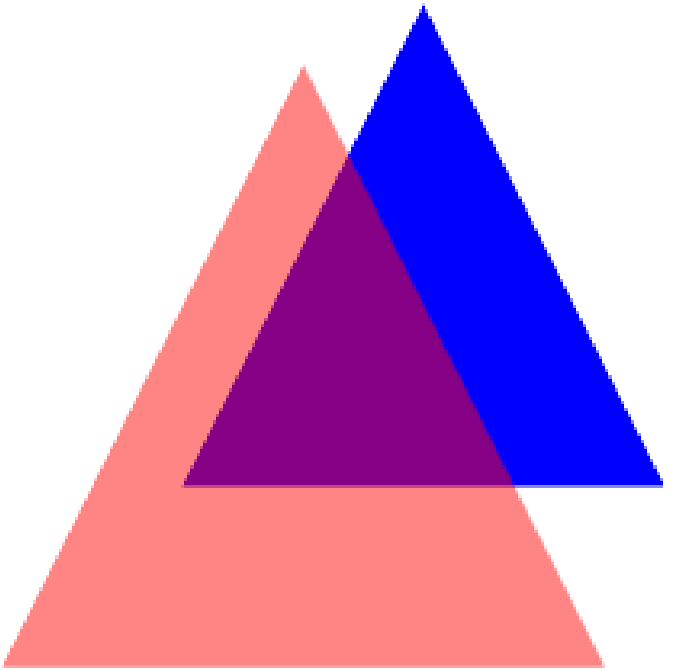
Constant	Relevant Factor	Computed Blend Factor
GL_ZERO	source or destination	(0, 0, 0, 0)
GL_ONE	source or destination	(1, 1, 1, 1)
GL_DST_COLOR	source	(R_d, G_d, B_d, A_d)
GL_SRC_COLOR	destination	(R_s, G_s, B_s, A_s)
GL_ONE_MINUS_DST_COLOR	source	$(1, 1, 1, 1)-(R_d, G_d, B_d, A_d)$
GL_ONE_MINUS_SRC_COLOR	destination	$(1, 1, 1, 1)-(R_s, G_s, B_s, A_s)$
GL_SRC_ALPHA	source or destination	(A_s, A_s, A_s, A_s)
GL_ONE_MINUS_SRC_ALPH A	source or destination	$(1, 1, 1, 1)-(A_s, A_s, A_s, A_s)$
GL_DST_ALPHA	source or destination	(A_d, A_d, A_d, A_d)
GL_ONE_MINUS_DST_ALPH A	source or destination	$(1, 1, 1, 1)-(A_d, A_d, A_d, A_d)$
GL_SRC_ALPHA_SATURATE	source	(f, f, f, 1); f=min(As, 1−Ad)

Constantes de combinación

- La transparencia se logra usando:

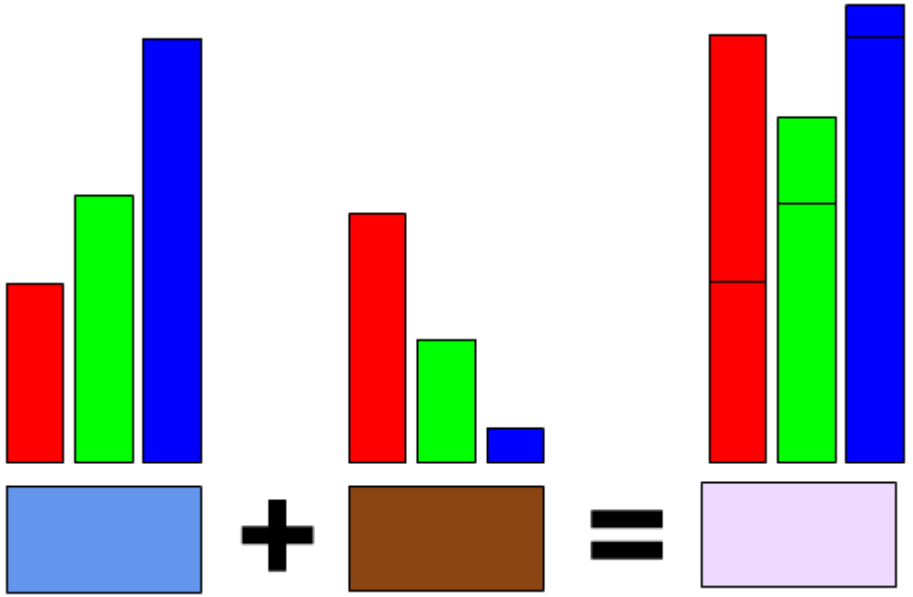
  `glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_ALPHA)`

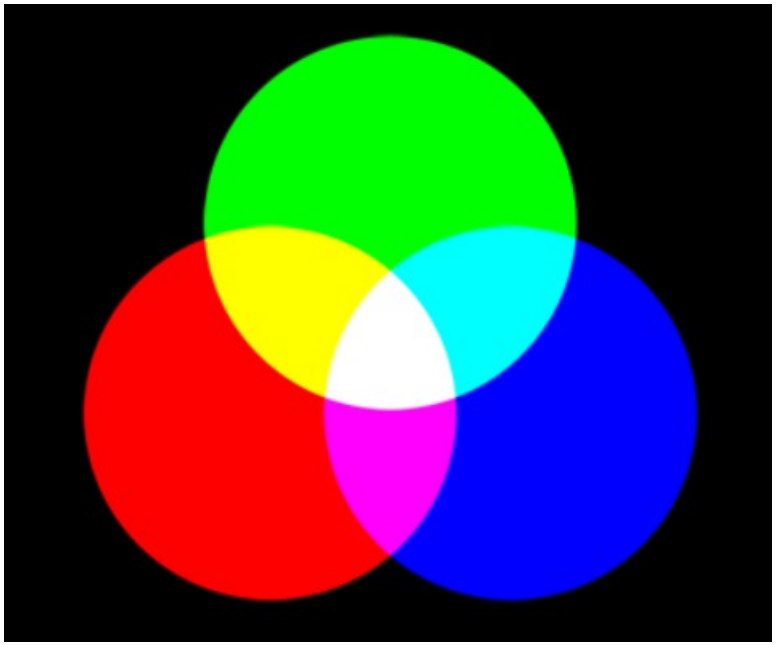
- Hay que dibujar primero las primitivas más lejanas y luego las más cercanas.



Additive Blending

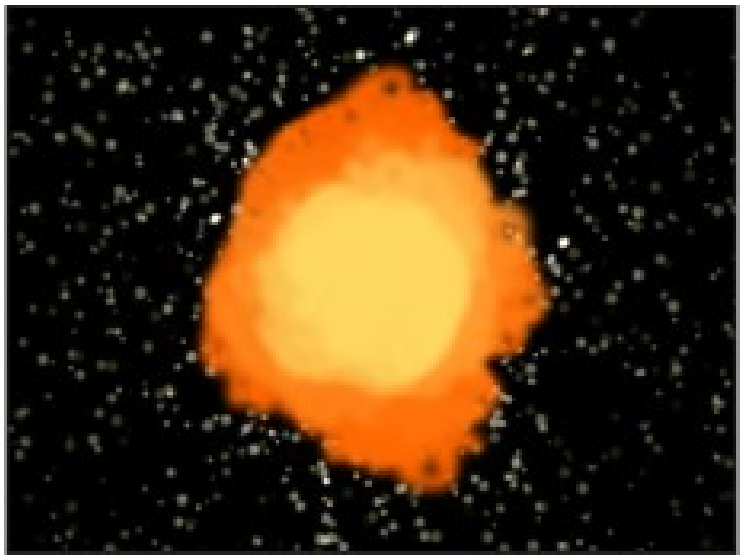
- `glBlendFunc(GL_ONE,GL_ONE)`
- Esta función de combinación es conmutativa.
- Los colores tienden a quedar más claros.



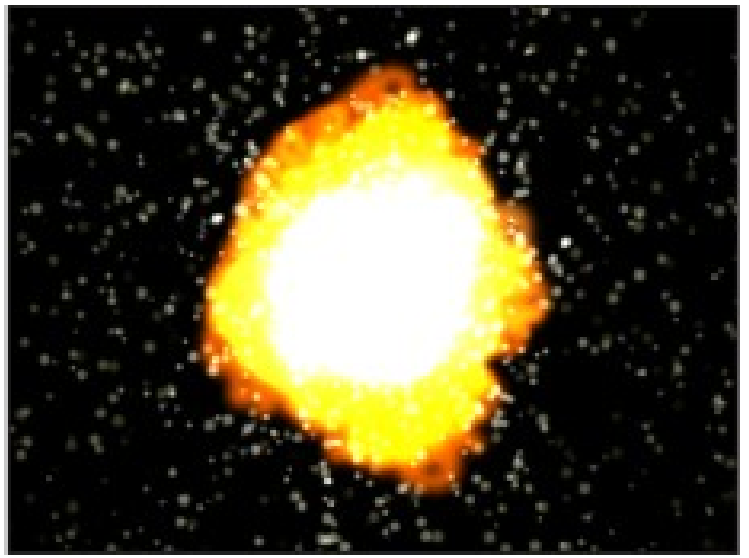


Additive Blending

- Se puede utilizar para dejar una textura más brillante.
- Se dibuja la primer vez sin *blending* y la segunda vez (la misma textura en el mismo lugar) activando este *blend*.



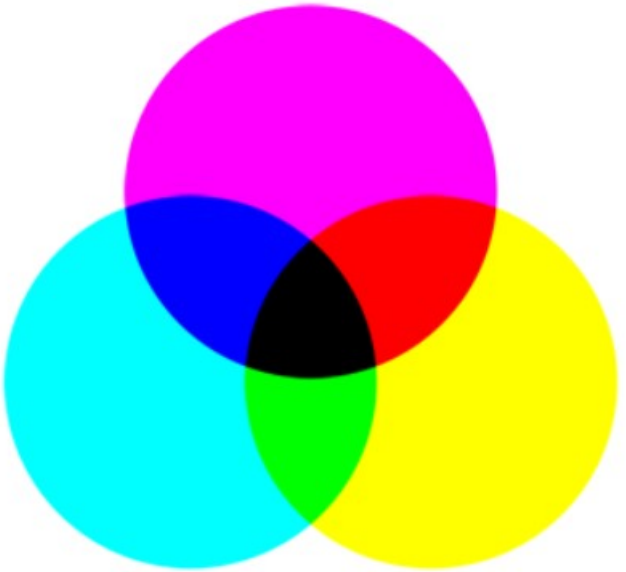
Additive Blend off



Additive Blend on

Subtractive Blending

- `glBlendFunc(GL_ZERO, GL_ONE_MINUS_SRC_COLOR)`
- Los colores tienden a quedar más oscuros

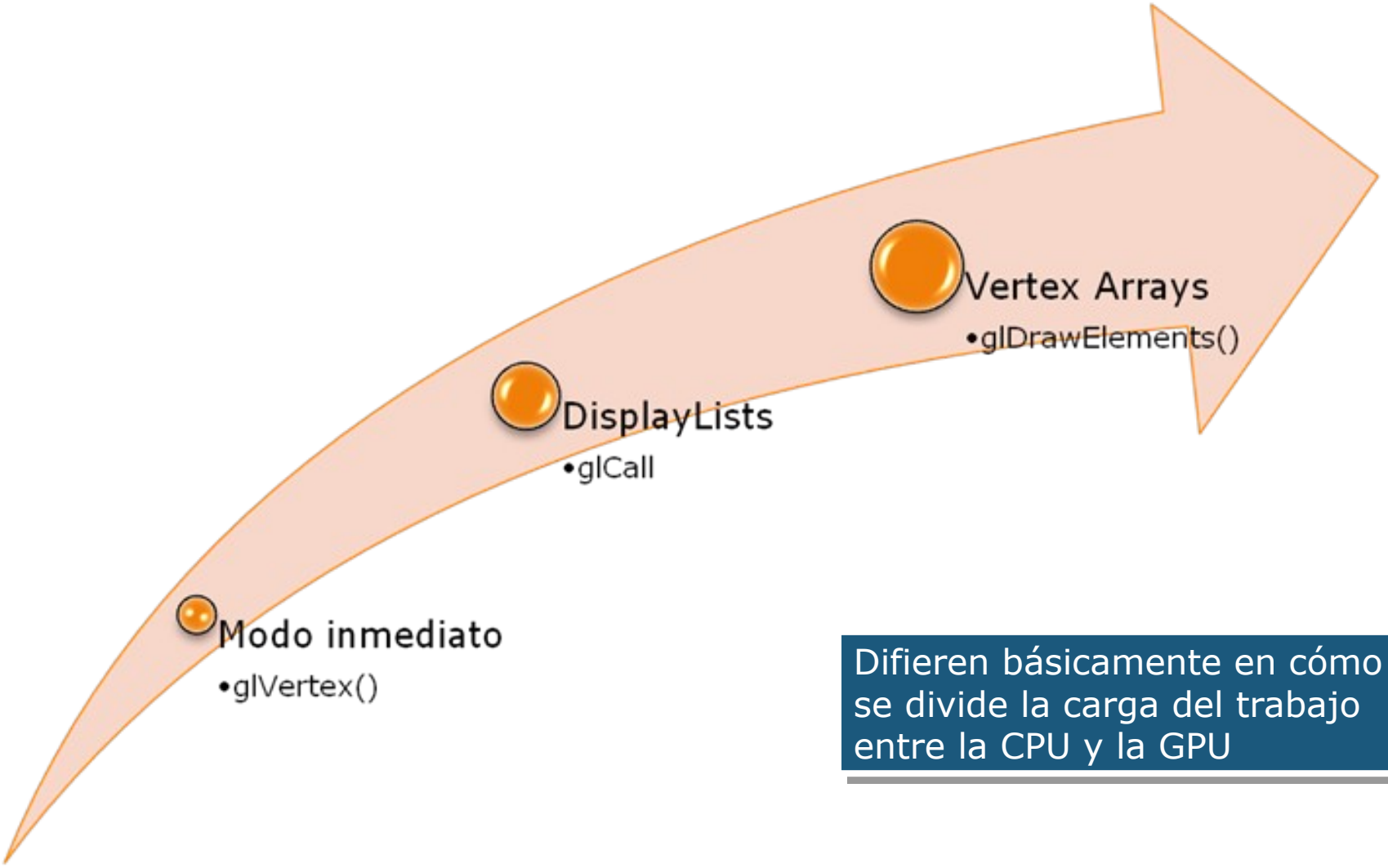


A tener en cuenta...

- Las transparencias no son compatibles con el Z-Buffer.
  - Los polígonos transparentes no ocluyen a los que tienen menor Z.
- Deshabilitar el Z-Buffer obliga a dibujar la geometría en orden.
- Una alternativa es:
  1. Dibujar primero la geometría opaca con el Z-Buffer activado
  2. Dibujar luego la geometría transparente con el Z-Buffer desactivado. Tener en cuenta el orden!

Uso eficiente del hardware

Modos de funcionamiento



Difieren básicamente en cómo se divide la carga del trabajo entre la CPU y la GPU

Modos de funcionamiento

- **Modo inmediato**
  - Las primitivas son enviadas al *pipeline* y desplegadas una a una.
  - No hay memoria de las entidades gráficas que pasaron.
- Ejemplo:

```
glTexCoord3f(...);
glVertex3f(...);      //primer vértice
glTexCoord3f(...);
glVertex3f(...);      //segundo vértice
```

Modos de funcionamiento

- **Display Lists**
  - Se puede almacenar una secuencia de comandos OpenGL para ser utilizados luego.
  - Las *Display Lists* se compilan y almacenan en el servidor gráfico (GPU)
  - Pueden ser desplegadas con diferentes estados (aplicado a toda la compilación)
  - Hay ciertos comandos OpenGL que no son compilados (ver la especificación de `glNewList`)

Modos de funcionamiento

- **Display Lists (cont.)**
- Ejemplo

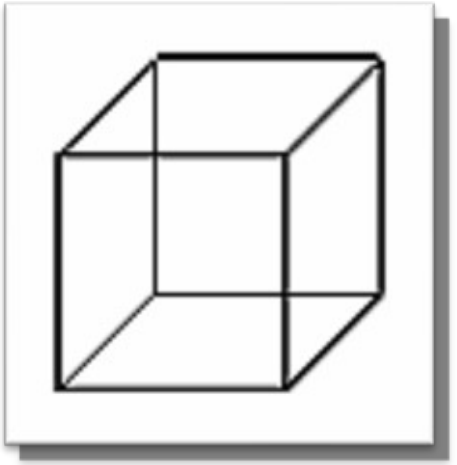
```
int displayList = glGenLists(1);
glNewList(listA, GL_COMPILE);
  glVertex3f(...);
  glVertex3f(...);
glEndList();

glCallList(displayList);   //dibujando
glTranslatef(...);
glCallList(displayList);   //dibujando
```

Modos de funcionamiento

- **Vertex Arrays**
  - Se pueden ejecutar muchas primitivas gráficas con una sola llamada a OpenGL.
  - Los datos se almacenan en *buffers* de estructura fija, lo que permite una gran optimización.
  - Los vertex arrays proveen mejor rendimiento de las aplicaciones porque formatean los datos para un mejor acceso de memoria.

Modos de funcionamiento

- **Vertex Arrays (cont.)**
  - Cuando se dibuja un cubo, se tiene que transmitir 3 veces la información de cada vértice



  - Usando *vertex arrays* se almacena la información de cada vértice sólo una vez en el hardware gráfico, que luego se la puede utilizar varias veces (menos invocaciones a funciones de OpenGL)

Modos de funcionamiento

- **Vertex Arrays (cont.)**
  - Se pueden manejar hasta 6 arrays con información
    - `VERTEX_ARRAY`
    - `COLOR_ARRAY`
    - `NORMAL_ARRAY`
    - `TEXTURE_COORD_ARRAY`
    - `EDGE_FLAG_ARRAY`
    - `INDEX_ARRAY`

Modos de funcionamiento

- **Vertex Arrays (cont.)**
- Ejemplo

```
GLfloat colores[24] = {...};  //8 colores
GLfloat vertices[24] = {...}; //8 vértices
GLubyte indices[24] = {...};  //6 quads

glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_VERTEX_ARRAY):
glColorPointer(3,GLFloat,0,colores);
glVertexPointer(3,GLFloat,0,vertices);
glDrawElements(GL_QUADS, 24,
                  GL_UNSIGNED_BYTE, indices);
```

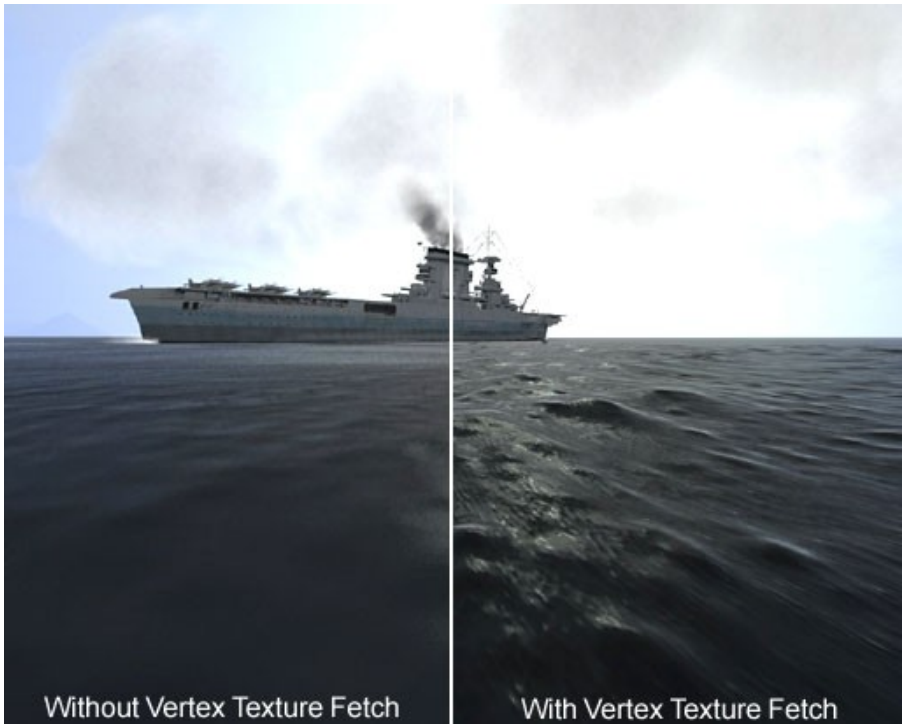
Temas avanzados

Shaders

- Sistema mediante el cual se puede asociar un programa a cada punto que es procesado por la tarjeta de video.
- Las primeras versiones eran muy limitadas en cuanto al tamaño del programa asociado.
- Las últimas versiones soportan loops y arrays.
- Permiten implementar el modelo de iluminación de Phong en tiempo real.

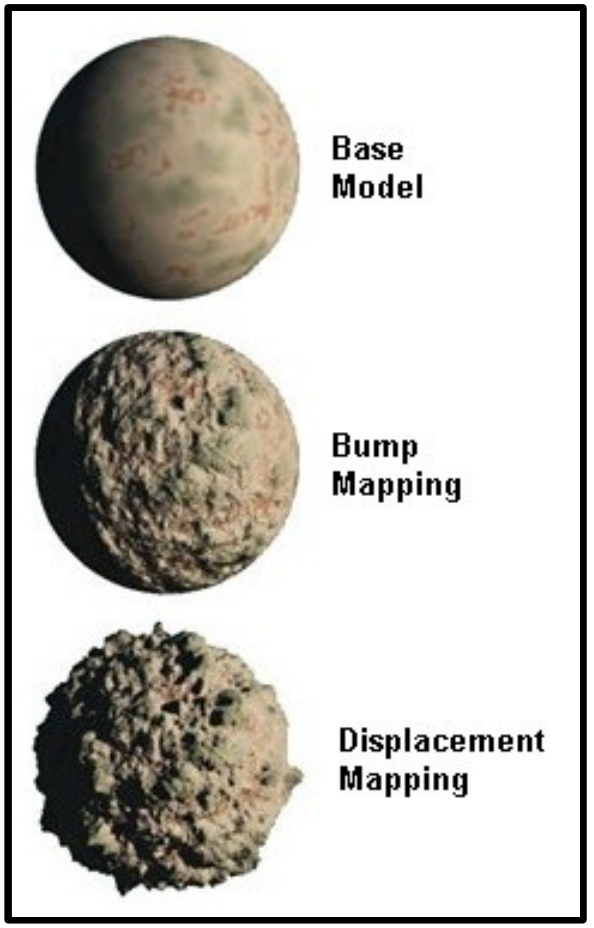
Ejemplos

- Generación de imágenes foto-realistas



Ejemplos

- Se puede aumentar la calidad del modelo sin aumentar la cantidad de polígonos.

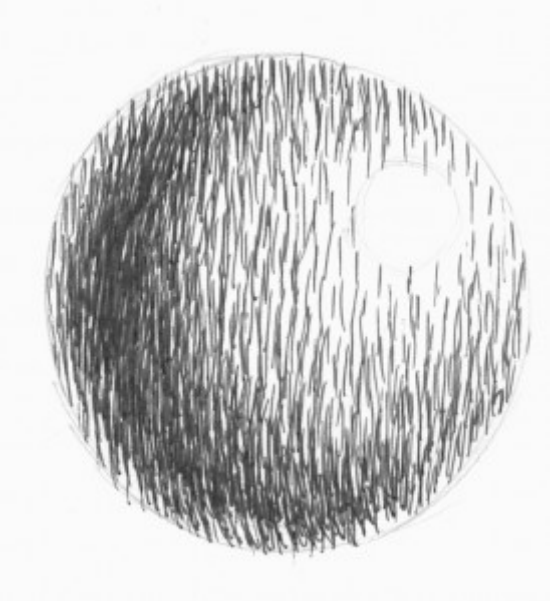


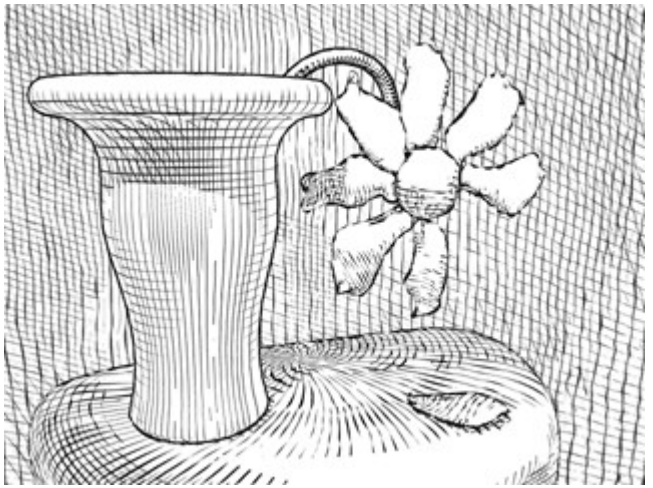
Ejemplos

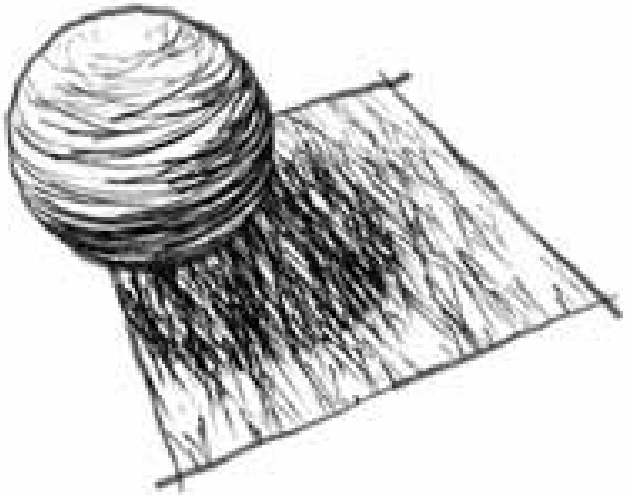
- Efectos no foto-realistas (cel shading)

Ejemplos

- Efectos no foto-realistas (hatching shader)







Documentación

- Especificación
  - http://www.opengl.org/documentation/specs/version1.5/glspec15.pdf
- OpenGL Reference Manual
  - http://www.opengl.org/documentation/blue_book/
- OpenGL Programming Guide
  - http://www.opengl.org/documentation/red_book/
- Otra información
  - http://www.opengl.org/
  - http://www.sgi.com/products/software/opengl/
  - http://www.sgi.com/products/software/opengl/examples/index.html
  - http://www.mesa3d.org/