Introducción al Blending

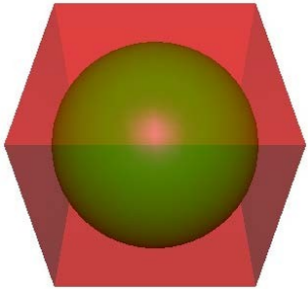
Informática Gráfica I

Material de: **Ana Gil Luezas**
Adaptado por: **Elena Gómez** y **Rubén Rubio**
{mariaelena.gomez,rubenrub}@ucm.es

Contenido

Colores RGBA

- El valor por defecto de la componente Alpha es 1 (≈ 255):
  Un color RGB se completa con $A = 1$ (opaco)
- Se utiliza para modelar objetos traslúcidos:
  $A = 1 \rightarrow$ opaco
  $A = 0 \rightarrow$ transparente
- Los colores de objetos que se superponen en la vista se combinan para determinar el color final: Cuando un objeto traslúcido aparece delante de otro.
- **Blending**: suma ponderada de los colores correspondientes a distintos objetos.
- **Alpha blending**: los factores de ponderación se determinan en función de la componente alpha.



Depth buffer y depth test

- El **Depth buffer** o **Z-buffer** contiene la distancia con respecto a la cámara (al plano cercano) de cada píxel (componente Z del fragmento). Los valores están en el rango [0, 1], siendo 0 el más cercano y 1 el más lejano.
- Inicialización:

```
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );
   // en IG1App::iniWinOpenGL()
glEnable(GL_DEPTH_TEST);       // en scene.init()
```

- Cada vez que se renderiza: `void display()`

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

- El back color buffer queda con el color de fondo en todos los píxeles, y el **Z-buffer** con el valor 1 en todos los píxeles.

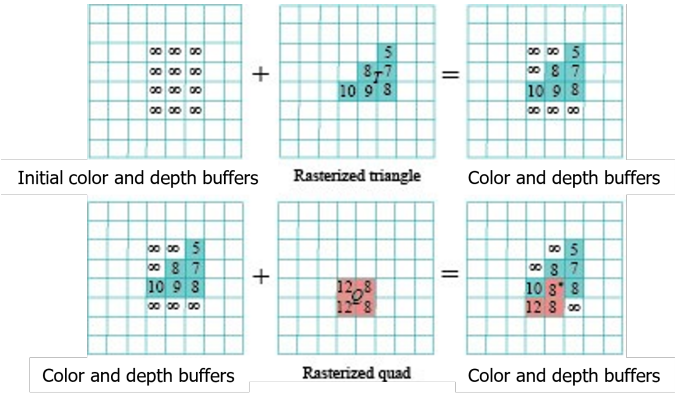
```
scene.render(camera); // ->
glutSwapBuffers();
```

Depth buffer y depth test

- Al realizarse la **proyección de los vértices**, se calcula la distancia relativa a la cámara y se guarda en la componente z del vértice.
- Se realiza el relleno de triángulos: genera los fragmentos del interior mediante interpolación de los valores de los vértices.
  Datos de cada fragmento: coordenadas de ventana (x, y, z) (proyectadas y ajustadas al puerto de vista), color (r, g, b, a), coordenadas de textura (s, t)
- Y se procesa cada fragmento:
  Depth Test por defecto `GL_LESS` (se puede configurar):
  - Cuando se procesa un fragmento, se compara la distancia del fragmento con el valor del Z-buffer.
  - Si es menor, el fragmento en proceso reemplaza el valor de ambos buffers (el de colores y el de profundidad).
  - En otro caso, el fragmento queda descartado y no modifica ningún buffer.

Color y depth buffers

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
T->render(...);        // triángulo azul opaco
Q->render(...);        // rectángulo rojo traslúcido
```

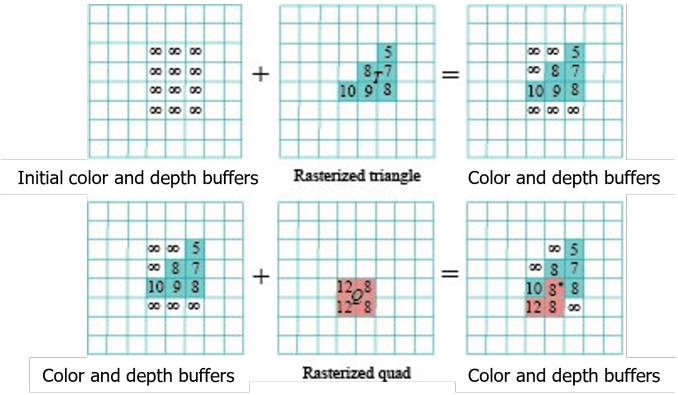


Con el test de profundidad por defecto activo y sin blending.

Un fragmento pasa el test si su profundidad es menor que la del buffer y sobrescribe ambos buffers con los valores del nuevo fragmento.

Color y depth buffers

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
T->render(...);        // triángulo azul opaco
Q->render(...);        // rectángulo rojo traslúcido
```



Con el test de profundidad por defecto activo y blending activo.

Un fragmento pasa el test si su profundidad es menor que la del buffer y **mezcla** el color del buffer con el color del fragmento. El valor del Z-buffer se reemplaza.

Ecuación de blending

- Activar y desactivar el Blending;

```
glEnable(GL_BLEND)
glDisable(GL_BLEND)
```

- Ecuación: la mezcla de los dos colores se obtiene con los factores de blending que estén establecidos

```
dstColor = srcBFactor * srcColor + dstBFactor * dstColor
```

siendo:

- `srcColor` = (`srcR`, `srcG`, `srcB`, `srcA`) el color RGBA del fragmento en proceso (source Color),
- `dstColor` = (`dstR`, `dstG`, `dstB`, `dstA`) el color del Color Buffer correspondiente al mismo píxel (destination Color)
- y `srcBFactor` y `dstBFactor` los correspondientes factores de blending.

Ecuación de blending

- Configuración de los factores de la ecuación:

```
dstColor = srcBFactor * srcColor + dstBFactor * dstColor
```

- El origen de los factores de la ecuación se puede establecer con `glBlendFunc(srcBFactor, dstBFactor)`:
  - `glBlendFunc(GL_ONE, GL_ZERO)` es el valor por defecto
  - `glBlendFunc(GL_CONSTANT_COLOR, GL_ONE_MINUS_CONSTANT_COLOR)` modula cada canal con un peso constante fijado con `glBlendColor(r, g, b, a)`
  - `glBlendFunc(GL_CONSTANT_ALPHA, GL_ONE_MINUS_CONSTANT_ALPHA)` solo usa el valor `a` para todos los canales

- **Alpha blending**: se utiliza la componente alfa de la malla que se renderiza:

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
```

Frame buffer

- El **Frame buffer** consta de varios buffers del mismo tamaño:
  1. **Colors buffers**: front and back
  2. **Depth buffer**: depth test
  3. **Stencil buffer**: stencil test
- Se configura al crear la ventana:

```
glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE |
                    GLUT_DEPTH | GLUT_STENCIL);
```

- Se reinician a los valores establecidos con:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |
        GL_STENCIL_BUFFER_BIT);
```

- **Los tests permiten descartar fragmentos** para que no aporten color al color buffer.
- Se pueden utilizar frame buffers auxiliares (frame buffer objects).

Frame buffer

1. Activar la escritura en el stencil buffer con glStencilOp.
2. Renderizar objetos específicos escribiendo solo en el stencil buffer.

3. Desactivar la escritura en el stencil buffer.
4. Renderizar objetos utilizando el contenido del stencil buffer con . `glEnable(GL_STENCIL_TEST)`.

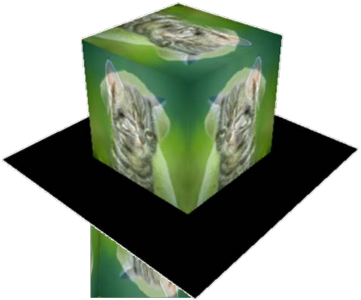
Stencil buffer

1. Renderizar el cubo.
2. Renderizar el suelo.
3. Renderizar el cubo invertido.

Problemas

- El depth test descarta los fragmentos tapados por el suelo.
- El reflejo aparece fuera del suelo.

Ejemplos: reflejos, sombras, perfiles



Stencil buffer

Ejemplos: reflejos, sombras, perfiles

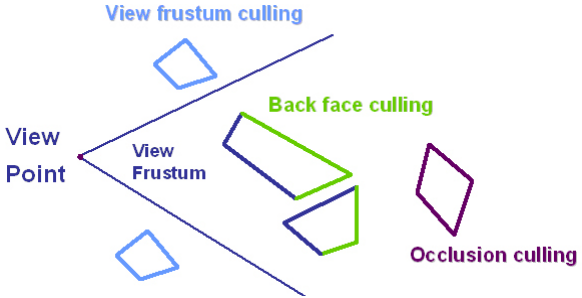
1. Renderizar el cubo de forma habitual.
2. Activar la escritura en el stencil buffer (valor 1).
3. Renderizar el suelo escribiendo en el stencil buffer y no en el Z-buffer.
4. Configurar el stencil test para que pasen los fragmentos con valor 1 en el Stencil buffer.
5. Renderizar el cubo invertido.
6. Desactivar el stencil buffer y configurar el Z-buffer para lectura/escritura.

Multitexturas

- Hay que utilizar tantas unidades de textura como imágenes queramos utilizar simultáneamente.
  1. Activar `glEnable(GL_MULTISAMPLE) //en scene::init`
  2. Cargar las texturas en la entidad que las utilice.
  3. Al renderizar la entidad activar tantas unidades de textura como texturas simultaneas utilice la entidad.
  4. Renderizar la malla activando las texturas en las unidades activas.
  5. Desactivar las unidades de textura.
- Es necesario enlazar las funciones de OpenGL posteriores a la versión 1.2.

Culling

- **Culling**: eliminar elementos, sacrificar selectivamente, entresacar o descartar
- Tipos principales de culling:
  - **Backface culling**: eliminación de caras traseras
  - **(view) Frustum culling**: eliminación de caras de una malla que están fuera del frustum
  - **Occlusion culling**: eliminación de objetos que están ocultos por otros objetos.



Back-face culling

- El back-face culling (o polygon culling) permite no pintar caras frontales, traseras o ambas
- Cuando una cara no se dibuja, se ve lo que hay detrás de ella
- El comando de OpenGL para hacerlo es `glCullFace(face)` donde `face` puede ser una de las constantes `GL_FRONT`, `GL_BACK` o `GL_FRONT_AND_BACK`
- El back-face culling se activa/desactiva con los comandos

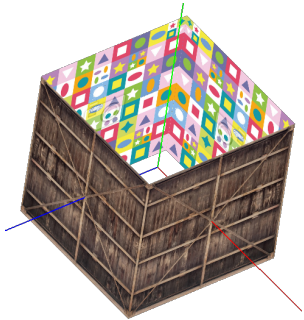
  ```
  glEnable(GL_CULL_FACE) / glDisable(GL_CULL_FACE)
  ```

- El culling permanece activado hasta que se desactiva expresamente
- Las escalas (negativas) afectan al culling

Texturas distintas en cada cara

- Se activa el culling con `glEnable(GL_CULL_FACE)`
- Se carga la textura de las caras frontales, se activa el culling de las caras traseras con `glCullFace(GL_BACK)` y se renderiza la malla

```
mFrontTexture->bind(GL_MODULATE);
glCullFace(GL_BACK)
mMesh->render();
mFrontTexture->unbind();
```



- Se repite lo mismo con la textura de las caras traseras y `glCullFace(GL_FRONT)`
- Se desactiva el culling con `glEnable(GL_CULL_FACE)`