Blending en OpenGL

Informática Gráfica I

Material de: **Ana Gil Luezas**

Adaptado por: **Elena Gómez** y **Rubén Rubio**

{mariaelena.gomez,rubenrub}@ucm.es

UNIVERSIDAD COMPLUTENSE MADRID

Contenido

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.

Depth Buffer y Depth Test

- Inicialización:

```
glEnable(GL_DEPTH_TEST);
```

- 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);
glfwSwapBuffers(mWindow);
```

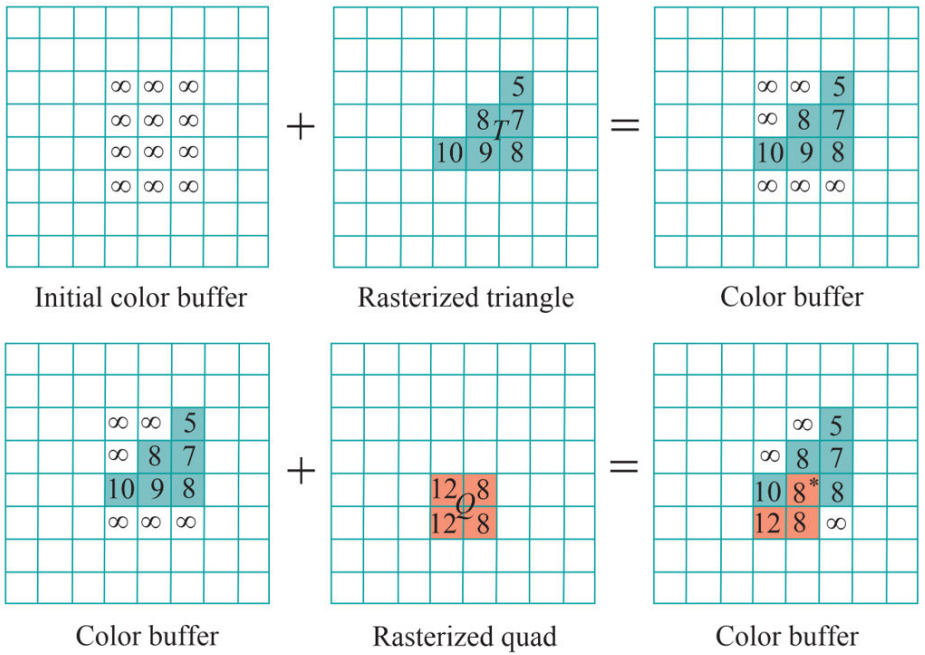
Depth Buffer y Depth Test

Depth Test por defecto `GL_LESS` (se puede configurar). Se procesa cada fragmento:

- 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
```

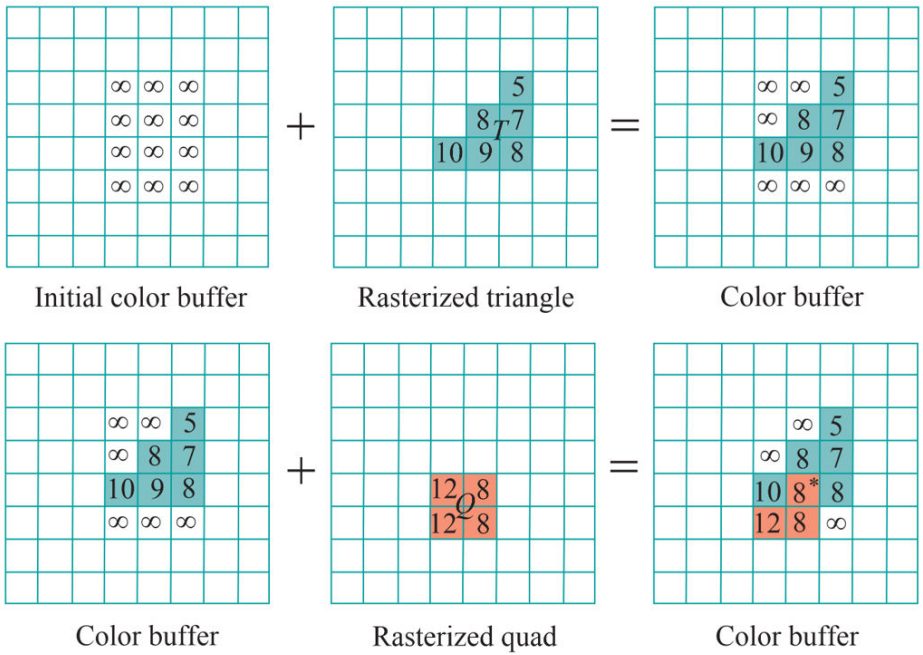


Con el test de profundidad por defecto activo y **sin blending**.

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)`:
  - Valor por defecto: `glBlendFunc(GL_ONE, GL_ZERO)`
  - Modula cada canal:

```
glBlendFunc(GL_CONSTANT_COLOR,
            GL_ONE_MINUS_CONSTANT_COLOR)
```

    con un peso constante fijado con `glBlendColor(r, g, b, a)`.
  - Único valor `a` para todos los canales:

```
glBlendFunc(GL_CONSTANT_ALPHA,
            GL_ONE_MINUS_CONSTANT_ALPHA)
```

Ecuación de blending

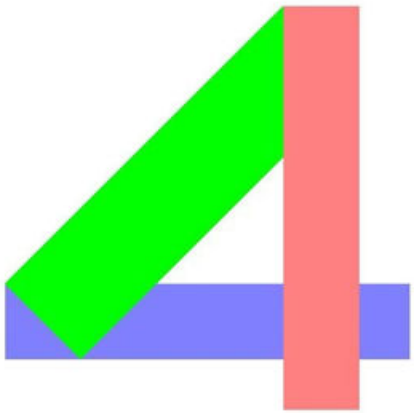
- **Alpha blending**: se utiliza la componente alfa de la malla que se renderiza:

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

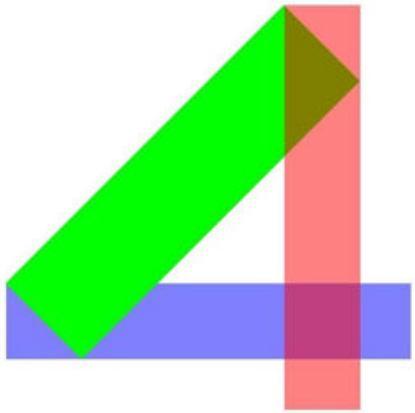
Alpha Blending

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

- Rectángulos: rojo traslúcido (cercano), azul traslúcido (lejano) y verde opaco (en medio).
  Con el test de profundidad por defecto y blending activos



$\rightarrow$



Orden: rojo, verde, azul

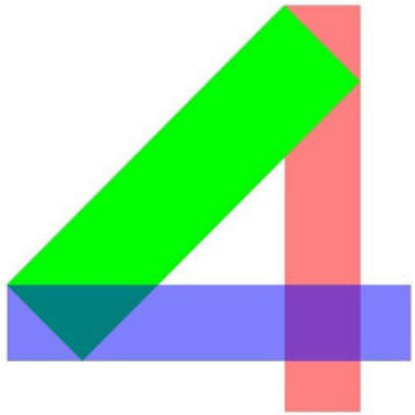
Orden: azul, verde, rojo

- La profundidad es relativa al punto de vista (posición de la cámara).

Alpha Blending

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

- Rectángulos: rojo traslúcido (cercano), azul traslúcido (lejano) y verde opaco (en medio).
  Test de profundidad desactivado y blending activo:



Orden: rojo, verde, azul

Alpha Blending

- Orden de renderizado con objetos opacos y traslúcidos:
  1. Test de profundidad activado por defecto.
     Dibujar los objetos opacos.
  2. Usar el buffer de profundidad solo para lectura:
     `glDepthMask(GL_FALSE);`
     realiza el test, pero no modifica el Z-Buffer:

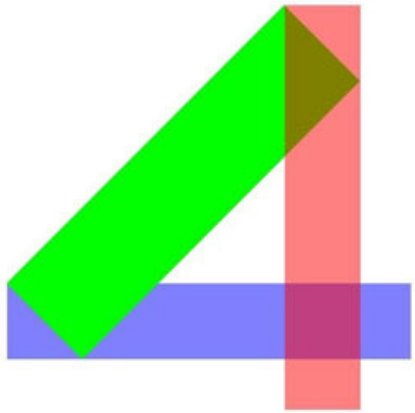
     ```
     glEnable(GL_BLEND);
     glBlendFunc...();
     ```

     Dibujar los objetos traslúcidos.
     Los que están delante de los opacos mezclarán el color.
  3. `glDepthMask(GL_TRUE);`
     `glDisable(GL_BLEND);`

Alpha Blending

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

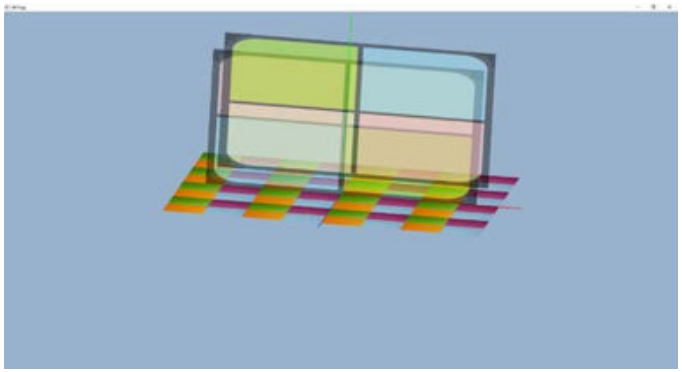
- Rectángulos: rojo traslúcido (cercano), azul traslúcido (lejano) y verde opaco (en medio).
  Test de profundidad activado para opacos y solo lectura para traslúcidos (y blending activo):



Orden (primero opacos):
verde, rojo, azul

Alpha Blending

- **Inconveniente**:
  - El orden en que los objetos traslúcidos hacen blending entre sí es importante (rojo sobre verde $\neq$ verde sobre rojo)
- **Solución**:
  1. Activar el blending y el z buffer solo para lectura.
  2. Ordenar los objetos traslúcidos de más alejados a más cercanos a la cámara.
  3. Renderizar los objetos traslúcidos con respecto a este orden.



Texturas con transparencia

Las texturas con fondo transparente pueden corresponder a objetos:

- **Opacos**: se renderizan sin blending y con el depth test habitual, pero para que el fondo no afecte al renderizado se descartan sus píxeles en el shader de fragmentos.
- **Traslúcidos**: se renderizan con blending y desactivando la escritura en el depth buffer. Se puede combinar también con lo anterior.

Texturas con transparencia

- El shader de fragmentos `texture_alpha` extiende `texture` con la siguiente instrucción

```
if (FragColor.a == 0.0)
    discard;
```

  que descarta el píxel (con la instrucción `discard`) si su canal alfa es nulo (transparente).
- En versiones anteriores de OpenGL se utilizaba el llamado *alpha test* (`GL_ALPHA_TEST`), que no está disponible en el perfil *core*.

Texturas traslúcidas

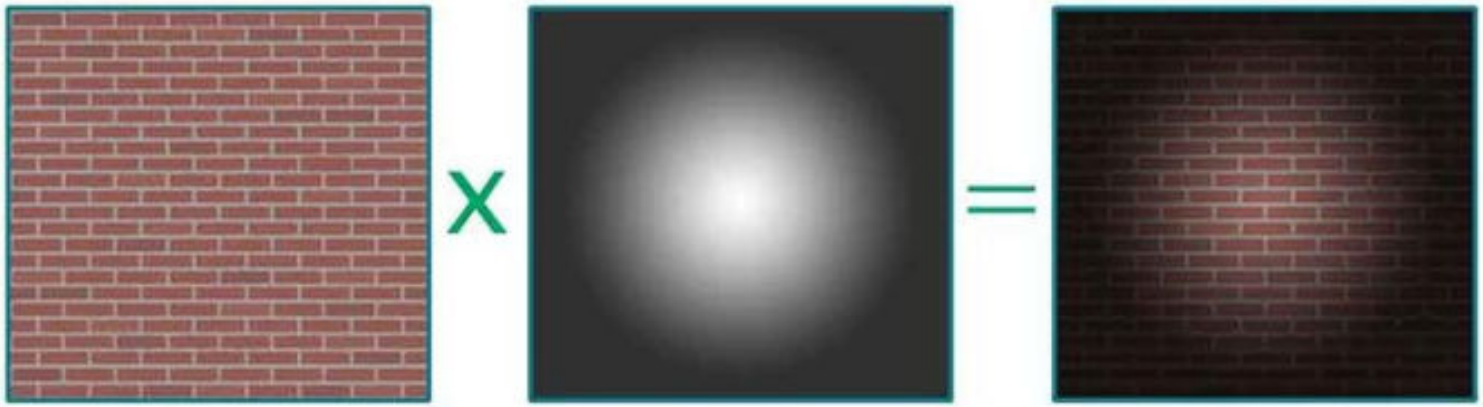
Los formatos BMP y JPEG no permiten representar imágenes con transparencia, pero el formato PNG sí lo permite. El método `load` de la clase `Texture` permite cargar imágenes en cualquiera de esos tres formatos:

```
void load(const string& filename, GLubyte alpha = 255);
```

Por defecto, si la imagen es RGBA (PNG), se mantiene el canal alfa de sus píxeles. En caso contrario se establece establece a opaco (255) en todos ellos. El argumento `alpha` permite especificar cualquier valor entre 0 y 255 para sobrescribir el canal alfa y hacer que la textura sea uniformemente traslúcida.

Multitexturas

- Se puede asociar más de una textura a un objeto, y combinar los colores de las imágenes para obtener el color del objeto.
- Hay que utilizar tantas unidades de textura como imágenes queramos utilizar simultáneamente.
- Por ejemplo: una imagen se mezcla con otra para añadir luz a la original, renderizando el rectángulo con dos unidades de textura activas.



Multitexturas

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
- GLFW configura un depth buffer y un stencil buffer por defecto, aunque en otras bibliotecas puede ser necesario crearlos expresamente al inicializar la ventana.
- 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)`.

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.

Back-face culling

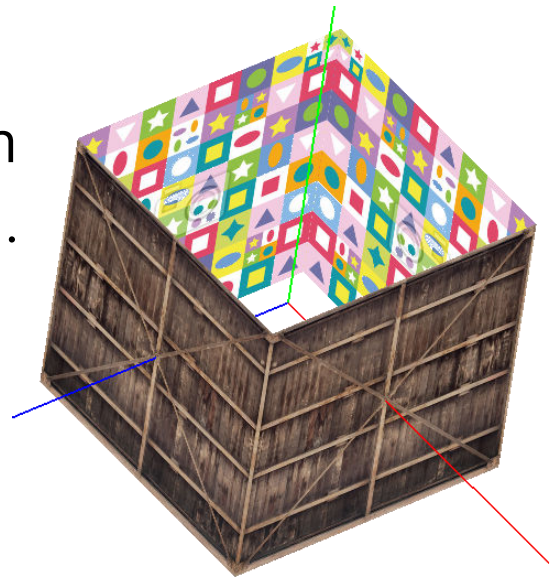
- El back-face culling (o polygon culling) permite no pintar caras frontales, traseras o ambas
- Cuando una cara no se dibuja, se ve lo que hay detrás de ella
- El comando de OpenGL para hacerlo es `glCullFace(face)` donde `face` puede ser una de las constantes `GL_FRONT`, `GL_BACK` o `GL_FRONT_AND_BACK`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