Some Visual Techniques in OpenGL

Blending

Colors at vertices etc. may have four channels.

RGBA = (red, green, blue, alpha)

Blending

Colors at vertices etc. may have four channels.

RGBA $=$ (red, green, blue, alpha)

What is the use of alpha?

Blending

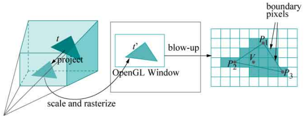
Colors at vertices etc. may have four channels.

RGBA = (red, green, blue, alpha)

What is the use of alpha?

Recall rasterization:

- Triangle vertices are projected to screen space.
- Pixels associated with triangle are found.
- Color value and z-value (depth) calculated for each (often using interpolation on vertex values, as well as texture look-ups).



Blending

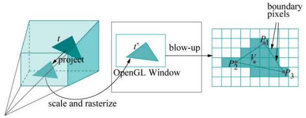
Colors at vertices etc. may have four channels.

RGBA = (red, green, blue, alpha)

What is the use of alpha?

Recall rasterization:

- Triangle vertices are projected to screen space.
- Pixels associated with triangle are found.
- Color value and z-value (depth) calculated for each (often using interpolation on vertex values, as well as texture look-ups).



Fragment = pixel coordinate + calculated color value and z-value. (A "potential" pixel in the final picture).

Blending

If passing various tests, e.g. the z-buffer test (more tests described later today), a fragment then normally overwrites a pixel in the framebuffer with its color value.

Blending

If passing various tests, e.g. the *z*-buffer test (more tests described later today), a fragment then normally overwrites a pixel in the framebuffer with its color value.

With blending, the color value of the pixel in the framebuffer instead becomes the weighted average between its old value and the value of the fragment.

Blending

If passing various tests, e.g. the *z*-buffer test (more tests described later today), a fragment then normally overwrites a pixel in the framebuffer with its color value.

With blending, the color value of the pixel in the framebuffer instead becomes the weighted average between its old value and the value of the fragment.

The weights are based on (usually) the alpha values of the fragments color.

Blending

If passing various tests, e.g. the z-buffer test (more tests described later today), a fragment then normally overwrites a pixel in the framebuffer with its color value.

With blending, the color value of the pixel in the framebuffer instead becomes the weighted average between its old value and the value of the fragment.

The weights are based on (usually) the alpha values of the fragments color.

Exactly how can be set in various way in OpenGL. Note: the averaging takes place individually on each of the color channels (including A-channel).

Blending Example

A typical example:

```
glEnable(GL_BLEND);
glBlendFunction(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

$$\text{Dest}_X = \text{Source}_{\text{alpha}} \cdot \text{Source}_X + (1 - \text{Source}_{\text{alpha}}) \cdot \text{Dest}_X$$

for X = red, green, blue, alpha. Dest is colorbuffer pixel value, Source is fragment value.

(All resulting channel values clamped to 1.0.)

Applications

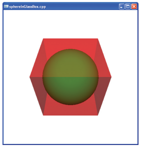
Blending useful for e.g.:

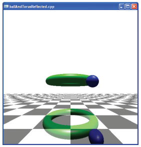
- Translucent objects.
- Reflections.
- Morphing between textures.
- Billboarding.

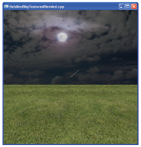
Applications

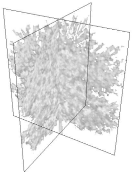
Blending useful for e.g.:

- Translucent objects.
- Reflections.
- Morphing between textures.
- Billboarding.







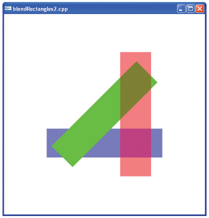


(From OpenGL Programming Guide)

Translucent Objects

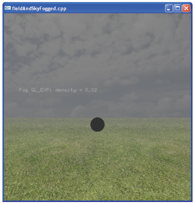
Drawing translucent objects:

- First draw all opaque objects (no blending)
- Then draw all translucent objects, with blending enabled, in back-to-front order wrt. the viewer [BSP trees may be used].



Fog

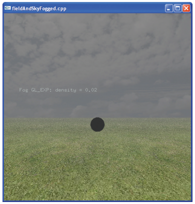
Automatic depth-based blending with fog-color.



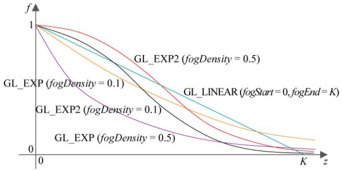
```
Fog GL_EXP: density = 0.02
```

Fog

Automatic depth-based blending with fog-color.

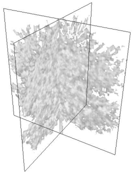


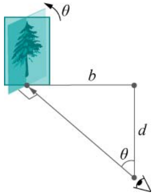
Various depth-functions can be chosen:



Billboards

A plain rectangle with a texture, simulating objects.



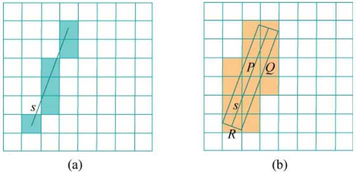


Useful for many things (especially when combined with translucent edges via blending), e.g.:

- Clouds
- Trees
- Laser beams
- Smoke
- Explosions

Anti-Aliasing

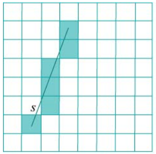
OpenGL may be asked to anti-aliase using blending.



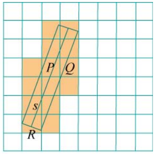
s

Anti-Aliasing

OpenGL may be asked to anti-aliase using blending.



(a)



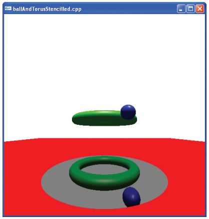
(b)

s

Another method is multisampling (several subpixels/rays per final pixel).

Further OpenGL Buffers

- Accumulation buffer: combining place for color buffer contents (entire pictures). Used for e.g. depth-of-field effects, motion blur effects.
- Stencil buffer: used to restrict drawings to various areas (think boolean bits per pixel - first set bits during a rendering to stencil buffer, then use bits during rendering to color buffer (stencil test)).



Picking

Picking $=$ select an object via mouse on screen.

Picking

Picking = select an object via mouse on screen.

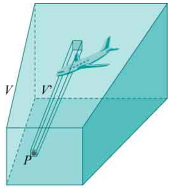
How?!?

Picking

Picking = select an object via mouse on screen.

How?!?

- Special render mode allows OpenGL to report on which rendered objects intersected the frustrum.
- You can name (number) the objects rendered.
- For each, OpenGL can report the min and max z-value inside the frustrum.
- GlU has command for setting up a pixes-wide frustrum.

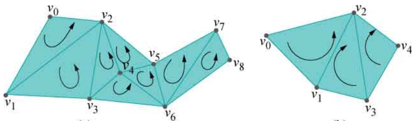


Viewing face

Orientation

Vertex order gives orientation on triangles [CCW side and CW side].

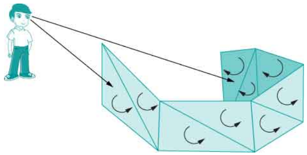
Several neighboring triangles: consistent orientation.

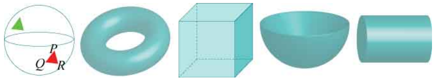


(a) (b)

Backface Culling

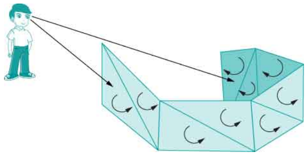
Save time by not shading backfacing triangles of closed objects.

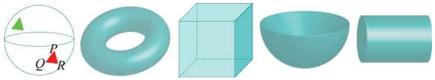




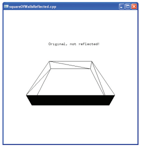
Backface Culling

Save time by not shading backfacing triangles of closed objects.

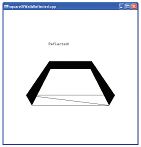




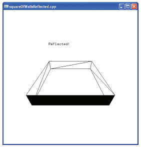
Note: reflections are orientation reversing transformations:



Original, not reflected!



Reflected!



Reflected!