

Aliasing

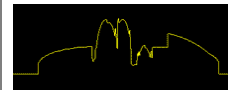
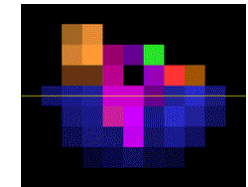
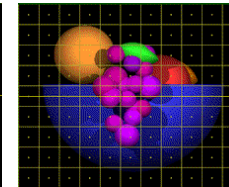
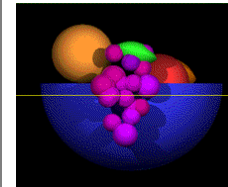
- A term from signal processing.
- Original application: frequency analysis.
- Problem: You have a signal that is continuous in time, but you "sample" (i.e. measure) the signal only at discrete points in time.
- If you don't sample often enough, you can misrepresent a high-frequency signal as a low-frequency signal.

Sampling a scene

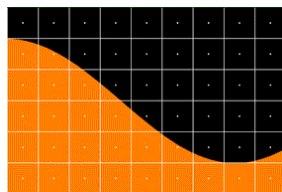
Original scene

Sampled scene

Rendered scene



Jagged Profiles

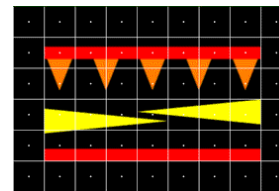


Original



Rendered

Loss of detail

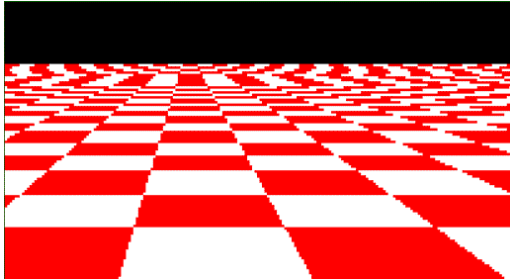


Original



Rendered

Disintegrating textures



Other Aliasing examples

- Waving hands in front of a TV.
- Strobe lights at discos.
- Video of wheels turning

Nyquist sampling theorem

- To avoid aliasing, need to sample at twice the highest possible frequency.
- Eg:
 - Audio CD's sample at 44.1KHz
 - Human hearing can only hear up to 20KHz.

Aliasing in graphics

- Similar concept, except not in terms of time, but in terms of space.
- Example: rather than "highest frequency" in Nyquist sampling theorem, can talk about smallest object.
- Otherwise, objects can "fall between" pixels.

Aliasing turns up in lots of places!

- Generic problem in computer graphics.
- Our monitors have discrete pixels ... can't switch on at an arbitrary point.
- Textures have limited resolution.
- Animations only generated at a certain number of frames per second.

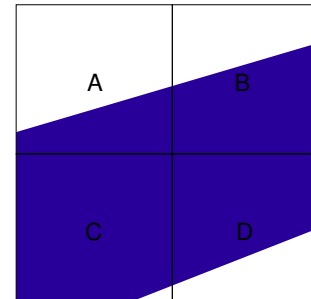
How to fix?

- One solution: Use higher resolution!
- Problem: Cost ... so how far can we stretch existing hardware?
- Antialiasing!
- 2 main approaches:
 - Prefiltering
 - Postfiltering

Prefiltering

- An approach which modifies our algorithm so that it takes into account the geometric properties.
- Instead of turning pixels on or off, work out how much of each pixel object covers.

Prefiltering example

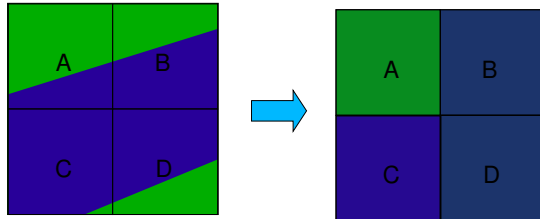


With no anti-aliasing, would fill C and D with pure blue, and A and B untouched

Prefiltering would work out ratio covered for each pixel.

Prefiltering might colour
A=0.2, B=0.7, C=0.95, D=0.7

Prefiltering continued



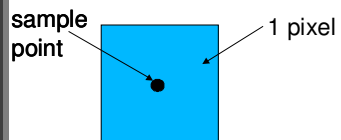
If the fraction of the pixel the line covers is a , the previous colour is P and the line colour is N , the resulting colour is the weighted average $aN + (1-a)P$

Prefiltering result

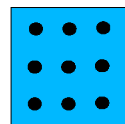


Postfiltering

- Rather than complicate calculations, just do the same calculations but for more samples.
- eg in raytracing



Normal raytracing
1 sample per pixel

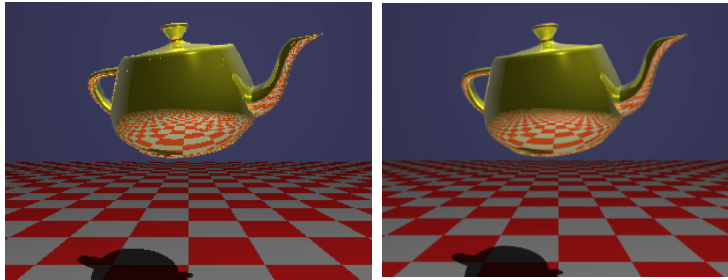


3x3 Antialiased raytracing
9 samples per pixel

Supersampling

- Taking more than one sample per pixel is called *supersampling*
- 3x3 supersampling means a total of 9 samples per pixel

Supersampling result



No anti-aliasing

3x3 super sampling

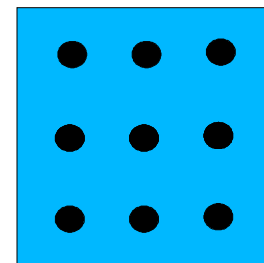
How to combine results?

- Can take an average of all the points.
- But better to give more weight to centre pixel.
- The weights we allocate to each of the pixels is called the *filter*.
- Equal weights for each sample point is called a *box filter*.
- If we want to smooth, can have overlapping boxes.

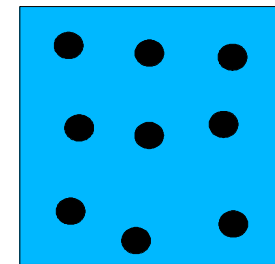
Problem: Still too ordered!

- If we sample in a grid, then this can cause other aliasing problems.
- Human eye "expects" noise. Image can be too perfect.
- So we randomly perturb the sample points to get rid of further aliasing effects.
- Officially called *stochastic sampling*.
- Usually called "jitter".

Jitter



Normal Supersampling



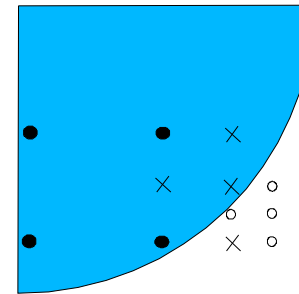
Jittered Supersampling

Adaptive supersampling

- Not all parts of the image need to be supersampled.
- Eg. Area of uniform colour doesn't change with supersampling.
- So supersampling is wasteful in some parts of image.
- Solution: Adaptive supersampling - only supersample when useful.

When is supersampling useful?

- When there is a large change in intensity!
- Can use a recursive approach



- = First pass
- × = Second pass
- = Third pass

Practical application

- All techniques are used.
- Postfiltering is frequently used in ray-tracing.
- Prefiltering is used in polygon rendering and line drawing. But so is postfiltering + supersampling.
- Modern graphics cards talk about FSAA = Full scene antialiasing.
- 2xFSAA means 2x2 postfilter antialiasing.

Antialiasing in OpenGL

- Prefilter antialiasing is supported.
- You can tell OpenGL to antialias lines & polygons.

```
glEnable (GL_LINE_SMOOTH);
```

- But you also must enable alpha blending

```
glEnable (GL_BLEND);
```

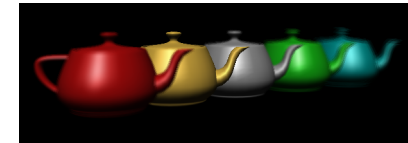
```
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Accumulation buffer

- Postfiltering also supported.
- Uses *accumulation buffer*.
- Accumulation is like normal frame buffer, but allows additions and averaging of images.
- So, render multiple images.
- OpenGL provides `accPerspective(...)` to do so. Basically moves the eye's position slightly ... by so-called "jitter" amount.

Other uses of accumulation buffer

- Accumulation buffer turns out to be useful for lots of applications.
- Motion-blur
- Depth-of-field



Demos

- OpenGL demo: aargb
- OpenGL demo: accanti
- OpenGL demo: dof
- POV-Ray results

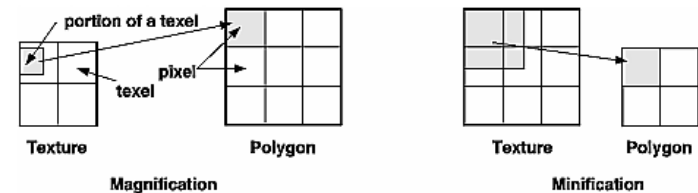
Antialiasing summary

- Types: Prefiltering, Postfiltering
 - Significantly improves image quality.
 - Gets rid of many visual artefacts.
- But:
 - Slow (both)
 - Memory usage (esp postfilter)
 - More complicated algorithm (esp prefilter)

Antialiasing and textures

- One area with aliasing is a real problem is textures.
- General problem: Texels and pixels are not going to be the same size.
- Three possibilities:
 - Magnification of texture: one texel maps to many pixels.
 - Minification of texture: many texels map to one pixel.

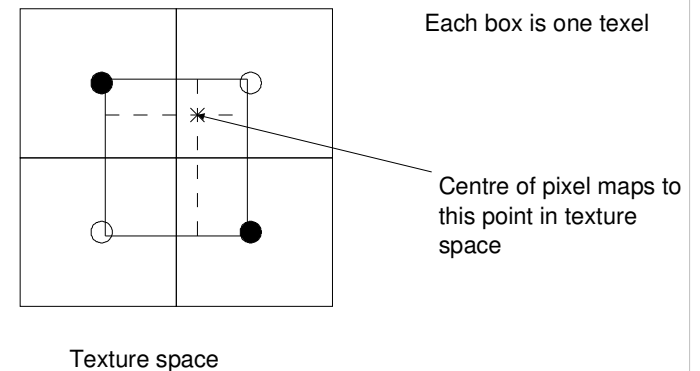
Magnification vs Minification



Solving the magnification problem

- Magnification leads to pixelation effects ... we can see the texels on the screen.
- How to fix?
- Linearly interpolate: Each intensity value is the centre of a pixel and then take a weighted average of surrounding four pixels.
- Called bilinear filtering.

Bilinear filtering



Comparison

- On previous slide, if using point sampling (GL_NEAREST), then it will be coloured white.
- If using bilinear filtering, it is a weighted sum of the four surrounding pixels, depending on distance.
- In this case, it would be coloured grey.

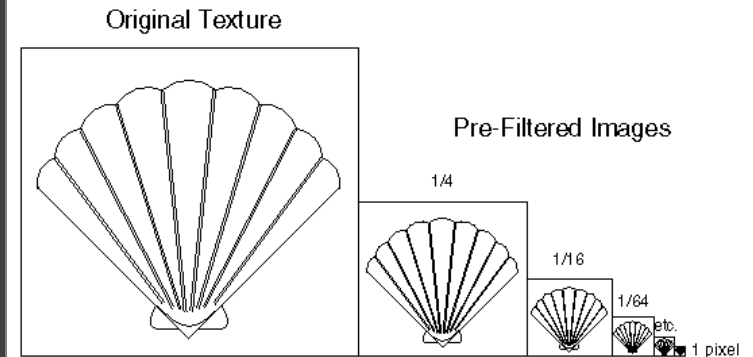
What about minification?

- Can also use linear interpolation, but what use is that?
- Point of using linear interpolation was to "smooth" edges and pixels.
- But real problem with min is the opposite.
- MIP = multim in parvum - "a lot in a small space".

MIPMaps.

- Say we have a 64x64 texture, then we calculate 32x32, 16x16, 8x8, 4x4, 2x2 and 1x1 ahead of time.
- Each is created by filtering higher resolution images down, so it is nicely rounded.
- Fits efficiently into memory.
- Depending on distance to object, we will use a different mipmap.

MIPMap diagram



MIPMap issues

- Idea: At different distances use the most appropriate MIPMap.
- Generally, the most appropriate is the one that gives the closest to 1:1 texel:pixel ratio.
- But can also interpolate between MIPMap levels.
- Called trilinear mipmapping.

Textures in OpenGL

Creating textures

- Steps:
 - Grab a name for the texture
`glGenTexture(number, textureName);`
 - Tell it we want to manipulate this texture
`glBindTexture(type, textureName);`
 - Set texture parameters
`glTexParameter(type, param, value);`
 - Load Texture from memory
`glTexImage2D(...)`

Texture parameters

- Several parameters are important:
- `GL_CLAMP` vs `GL_REPEAT` controls repetition.
- Defined separately for s (horizontal) and t (vertical).
- How to interpolate?

Interpolation controls

- Separate controls in OpenGL for what to do when magnifying and minifying textures.
- Options (for now): GL_NEAREST - point sampling, GL_LINEAR - bilinear interpolation.
- e.g. `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);`

glTexImage2D

- level of texture
- type of texture
- internal format
- width
- height
- border
- data format
- data type
- data

Using a texture.

- Enable texturing:
`glEnable(GL_TEXTURE_2D);`
- Tell it how to use texture:
`glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);`
- Tell it which texture to use:
`glBindTexture(GL_TEXTURE_2D, texName);`
- Specify texture coordinates between `glBegin` and `glEnd` statements using `glTexCoord2f(s,t)`.

Different modes

- GL_REPLACE: Don't do lighting.
- GL_MODULATE: Still use lighting, use texture to modulate kd of the surface.
- Two other modes: GL_BLEND and GL_DECAL, but don't worry about them for now.

Using MIPMaps

- This is what the level parameter was for.
- But for us, easier to use gluBuild2DMipmap
- gluBuild2DMipmaps(type, internal format, width, height, format, data type, data)
- Builds all the mipmaps for us.
- See JOGL TextureIO class.
- Demos – mipmap, texture

Animation

- So far, really focused on generation of static images.
- So ... how do we make moving images?
- Answer: make static images very fast.

Human perception

- Terminology: fps = frames per second. Hertz (Hz) is sometimes also used.
- "Magic" 20fps.
- Different parts of eye have different sensitivity - centre has low temporal sensitivity, high spatial sensitivity, edge of vision is opposite.
- Can improve up to maybe 80Hz at most ... but mostly to do with stability of image.

Double buffering

- Hence way to animate is to draw new image into frame buffer.
- Problem: If this happens, then the user sees us drawing new image. Flickering!
- So, draw in an off screen buffer, then tell hardware to switch buffers.
- Then use the buffer we were using before and then swap back.

Double buffering

- Pro: Makes MUCH better animations.
- Con: Takes up more memory
- Can also talk about triple buffering etc.
- Why? To smooth out processor load variations. But introduces lag.

Lag

- Usually only a problem in interactive situations.
- Difference in time between a change in input and a change in the output image.
- E.g. move mouse, scene changes ½ second later.
- When using HMD's has been known to cause "VR sickness".

Implementation in OpenGL

- Not handled directly by OpenGL, but by its companion library, e.g. GLUT or JOGL
GLComponent (glc) and GLCapabilities (cap).
- To switch on double buffering:
 - cap.setDoubleBuffered(DOUBLE_BUFFERED).
 - To swap buffers: glc.repaint()

JOGL Gory Details

- Two ways to use JOGL (and similarly for GLUT):
- Continuous rendering mode:

```
Animator animator = new FPSAnimator(glc, 30);
animator.start();
```
- Will call glc.repaint() 30 times a second, which will call your display() method.
- On-demand rendering mode: Screen is only repainted when glc.repaint() is explicitly called.

On-demand vs continuous

- Continuous sucks up lots of CPU time. On-demand doesn't.
- Assignment 1 uses double buffering and on-demand rendering.
- Assignment 2 uses double buffering and continuous rendering
- Why?
- What are disadvantages?

Dodgy problem with assignment

- If we test assignment on fast machine, your car will go fast, on a slow machine, it will go slow.
- Why? Because our updates to car position are on a frame basis and a slow machine won't do 30 fps.
- What we want: on a faster computer, your car moves more smoothly, on a slower computer, less smoothly, but at same speed.

Animation timing

- Best way: run physics and graphics as separate threads.
- Physics thread runs at a fixed rate (e.g. 60Hz)
- Graphics thread runs as fast as possible, but uses latest physics information.
- Hack solution: Check how much time has elapsed since last render, and move car by that amount.

Modelling for animation

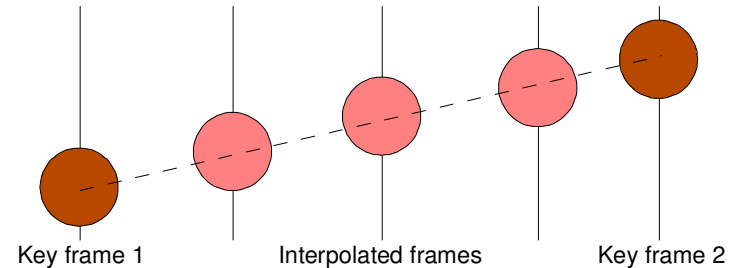
- How do you animate? What modelling techniques?
- Can come from physics. Example: Your assignment.
- But what if we have to work out data, e.g. modelling character?
- Key frame interpolation is easiest way.

Keyframes?

- Consider movement.
- Example: walking.
- I could specify two key frames: foot on ground and foot at peak.
- For each key frame, I specify vertex locations
- Then interpolate between the start and end frames.

Tweening

- Tweening is the process of creating intermediate images between key frames.
- Example



Key frame interpolation

- This is linear interpolation, but looks kind of unnatural.
- So can use "smoother" types of interpolation between points, e.g. fit a cubic to the points.

Physics modelling

- Using physics to work out position of objects.
- Can get very complicated, e.g. full-blown collision detection and interaction.
- Can even model gravity etc.
- Example: ODE (Open Dynamics Engine)

Skeletal animation

- Problem with both approaches: They function on vertices. But not very realistic model, because it's not polygons that move, but bones and structure.
- Solution: Do keyframe interpolation and physics on the bone structure, and "skin" the skeleton with polygons.
- This is called skeletal animation. The process of adding vertices is called "vertex skinning".

Level of detail

- Problem: When rendering a big world, there are too many polygons.
- Many polygons only map to a few pixels anyway.
- For fixed environments, use simpler model.
- But what if user can move around, zoom in and out etc?
- Solution: LOD control

LOD Approach

- If object is far away, use very simple model
- As we get closer, use more complicated in-depth model.
- E.g. Building might first be modelled as texture-mapped cube. Then as we get closer, this is replaced with geometry instead of texture maps.