

# OpenGL Effects

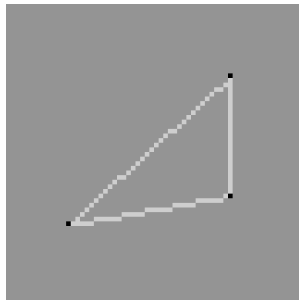
COMP 3003 Autumn 2005

# OpenGL Effects

- Antialiasing (spatial and temporal)
- Depth of focus, fog
- Polygon Offset
- Textual Information
- Selection & Picking
- Shadows

# Aliasing

- Aka the *jaggies*
- Not enough pixels
- Eye isn't fooled
- Worse if moving
  - the *crawlies*



# Use More Pixels



# Why this Happens

- Retinal cells see small patches
  - integrate incoming light on that patch
- Pixels are *not* integrated
  - sampled at *one* point
- Solution: average several samples

# Another Explanation

- Pixel-sized patches are larger than cells
- And retina is good at *edge* detection
- So we perceive the edge of the pixel
  - our eyes work against us
- Solution: *blur* the pixel

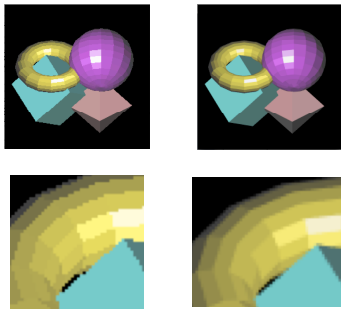
## Technical Explanation

- Eye is *reconstructing* image internally
- The edge is a high-frequency object
  - hard to reconstruct
  - needs more samples (pixels)
- All of these boil down to this:
  - we need more samples

## Averaging Samples

- Use the accumulation buffer
  - render the image several times
  - *jitter* the camera (move it slightly)
  - each image is *slightly different*
  - edge is no longer so abrupt

## Red Book Example



Aliased

Anti-Aliased

## Multisampling

- We could sample more pixels than we show
  - E.g. sample 1600x1200, display 800x600
  - Each pixel shown is average of 2x2 samples
  - This is called *multisampling*
  - Also called *full-screen anti-aliasing (FSAA)*
  - How you jitter the pixels is important

## Temporal Aliasing

- What happens to a fast moving object?
  - Our eyes integrate light over *time*
  - We see the *sum* of its positions
  - If it's too fast, it's *blurred*
- We do this in the accumulation buffer

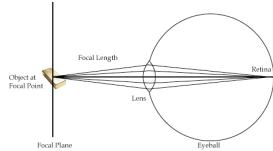
## Motion Blurring

- Render several times
- Move object each time
- Accumulate frames
- And voila!



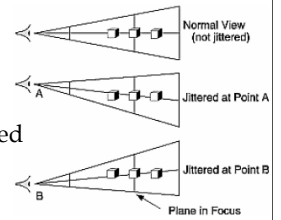
## Depth of Focus

- Our eyes focus at a fixed z-distance
  - the *focal distance* or *depth of focus*
  - objects at the focal distance are sharp
  - objects at other distances are not
    - they're *blurred*



## Accumulating Depth of Focus

- Jitter camera slightly
- Keep focal plane fixed:
  - objects in plane are fixed
  - other objects move



## Accumulated Result

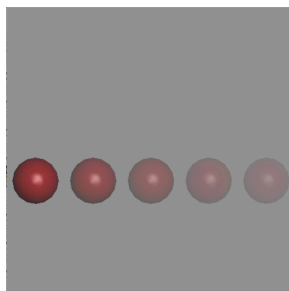


## Fog / Haze

- Light in a ray can get *scattered*
  - reflected from microscopic particles
  - fog (water), smoke (soot), haze (dust)
- Objects therefore *fade* with distance
  - uses depth from the depth buffer
  - `glEnable(GL_FOG)`, &c.

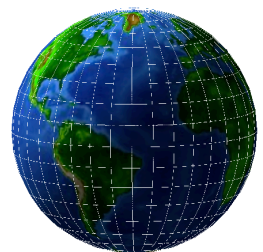
## Example of Fog

- Details are in the Red Book



## Polygon Offset

- What if we *want* to draw triangle edges?
- Draw once as a solid, once as wireframe
- But we run into *depth buffer quantization*



## Bigger Wireframe

- Scale the wireframe
- Easy for sphere
- Doesn't always work nicely
- Lines don't *quite* match triangles

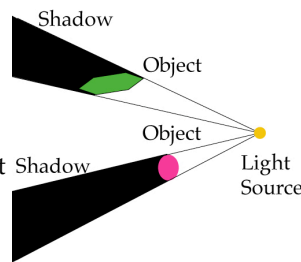


## Polygon Offset

- Or get OpenGL to take care of it
- `glEnable(GL_POLYGON_OFFSET);`
- `glPolygonOffset()` moves the polygon
  - slightly towards or away from eye
  - gets rid of the problem
- useful for decals & hidden lines

## Shadows

- What is a *shadow*?
- It's not an object
- It's an *absence* of light
  - behind a solid object
- Important depth cue
- How do we do this?

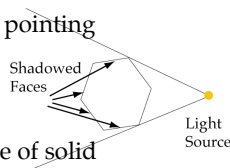


## Front & Back Sides

- Every triangle has a *front* and a *back*
  - normal vector points out from *front*
- For a solid object, the *back* is inside
  - therefore light never reaches it
  - so OpenGL doesn't bother with it
  - there's shadow *inside*

## Shadowed Faces

- *Shadowed faces* have normals pointing *away* from light source
- So  $\vec{n} \cdot \vec{v}$  is negative
- Light is blocked by other face of solid
  - no diffuse or specular light
  - just ambient and emissive

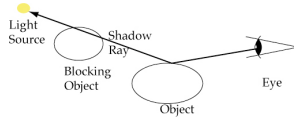


## Shadow Regions

- Compute shadows geometrically
  - which regions of space are shadowed
  - very expensive computationally
  - you can *precompute* this
    - for static objects only
    - apply as a texture

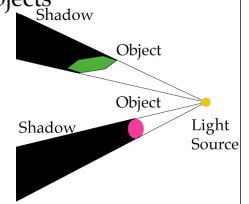
## Shadow Rays

- When raytracer hits a surface
- Draw a ray towards the light
- If it hits anything else, it's shadowed



## Projected Shadows

- Shadows are *projections* of objects
- Projected *away* from light
  - instead of *towards* eye
- We know how to do this!
- we've seen it before



## Projected Shadows

- Project vertices of *object* to a plane
  - i.e. draw *projection* of object on plane
  - apply a *projection matrix* to model
- Compute separately for each surface
  - Mostly only compute for the floor

## Shadow Projection

- Render the object
- Turn off lighting
- Set colour to black
- Apply projection
- Render again



## Circular Shadows

- For projected shadows, we render twice
- Older cards couldn't afford this
- So they just render a circular shadow
- Always underneath the character
- Better than nothing

## Shadow Maps

- Another way to exploit shadow projection
- Render image from viewpoint of light
- Anything visible is lit
- Anything else is shadowed
- Store result in texture & use in next pass

## Soft Shadows

- *Soft shadows* are caused by multiple lights
  - Triangle may not be shadowed by all
- So use the accumulation buffer
  - Render with shadows from each light

## Moving Light Sources

- Light position specified in *world* coordinates
- Apply transformations to move light around
- Use *glPush()* and *glPop()* so model isn't moved
- Or move light with model

