University of BRISTOL

COMS 30115

(Texture) Mapping

Carl Henrik Ek - carlhenrik.ek@bristol.ac.uk

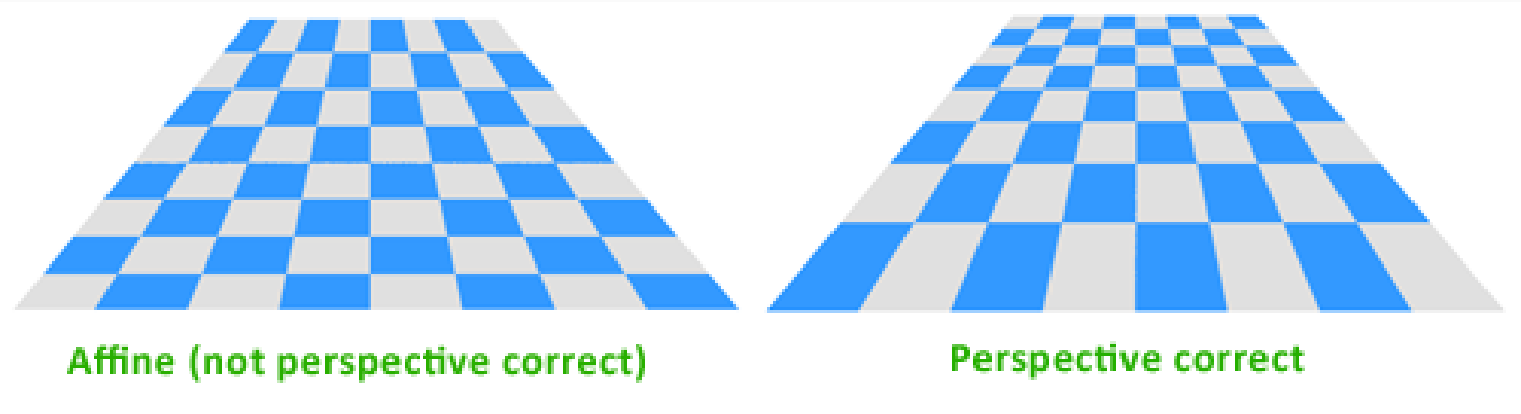
March 9th, 2018

http://www.carlhenrik.com

Last time

- Perpective correct interpolation
- Interpolation is the key to speed-up rendering
- Vertex shading vs. Pixel shading
- Interpolating attributes
    - Depth buffer

Perspective Correct



$$z(q) = \frac{1}{\frac{1}{z_0}(1-q) + \frac{1}{z_1}q}$$

$$c(q) = z\left(\frac{c_0}{z_0}(1-q) + \frac{c_1}{z_1}q\right)$$

Today

- Quantities to interpolate
  - Textures
- Buffers
  - stencil buffer
  - accumulation buffer
- Shadows

The Book

There is sadly not much in the book to read about this

Texture Mapping

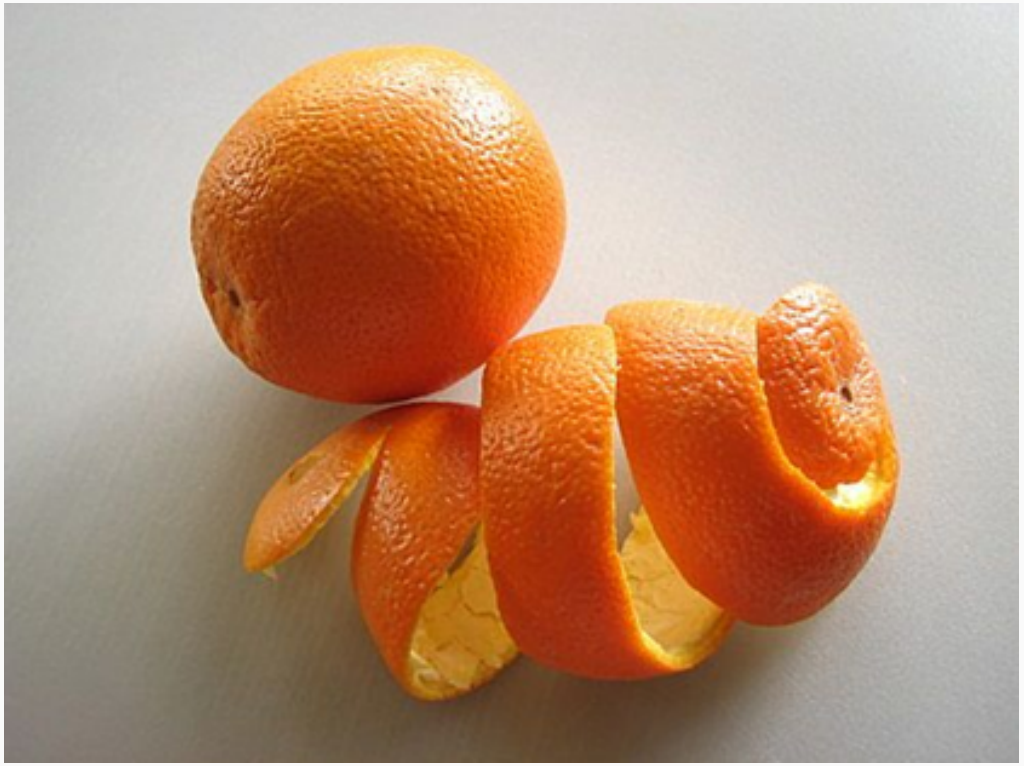
Texture Mapping

- Many ways to think about it
- Not just image, think about any quantity/surface property that you do not want or can interpolate
- Map a buffer across a polygon to use inside the pixelshader
- However, normally we see it just as an image
- But that is actually not completely straight forward

Texture Mapping

Texture Mapping

Texture Mapping



Gaussian Curvature

Theorema Egregium (The remarkable theorem)[1]

"The gaussian curvature of a surface is invariant to the local geometry of the surface"

- bending does not change the local geometry
- stretching does
- the above is a very important result in differiental geometry

[1] https://en.wikipedia.org/wiki/Theorema_Egregium

Gaussian Curvature

Theorema Egregium (The remarkable theorem)[1]

"The gaussian curvature of a surface is invariant to the local geometry of the surface"

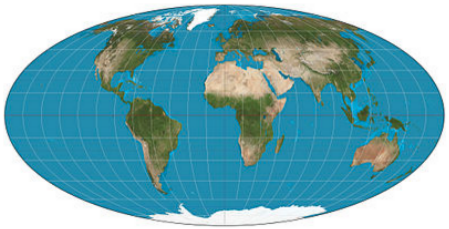
- bending does not change the local geometry
- stretching does
- the above is a very important result in differiental geometry
- *take home message, its not completely obvious to map textures*

[1] https://en.wikipedia.org/wiki/Theorema_Egregium

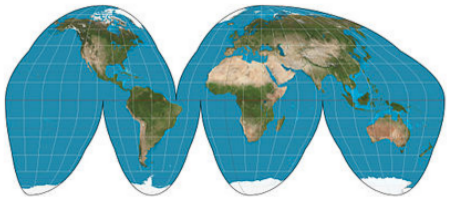
Maps

- Tobler hyperelliptical

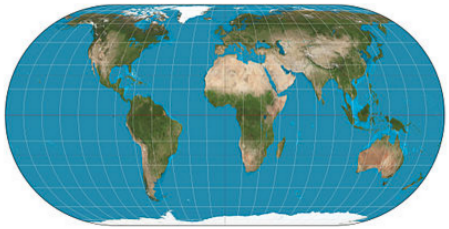
- Mollweide



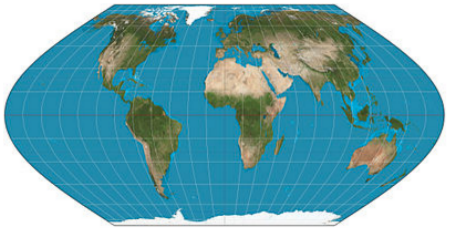
- Goode homolosine



- Eckert IV

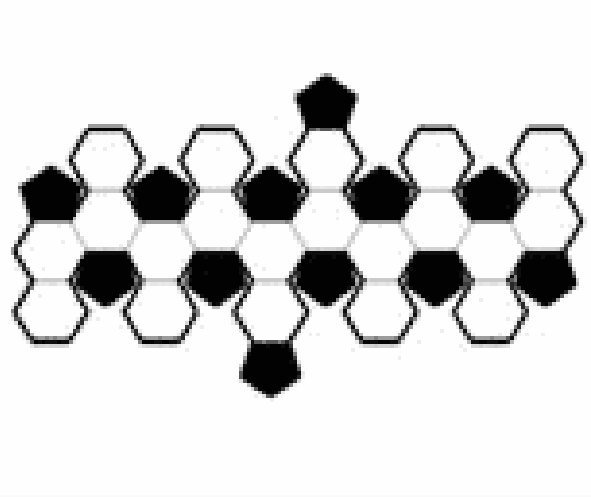


- Eckert VI



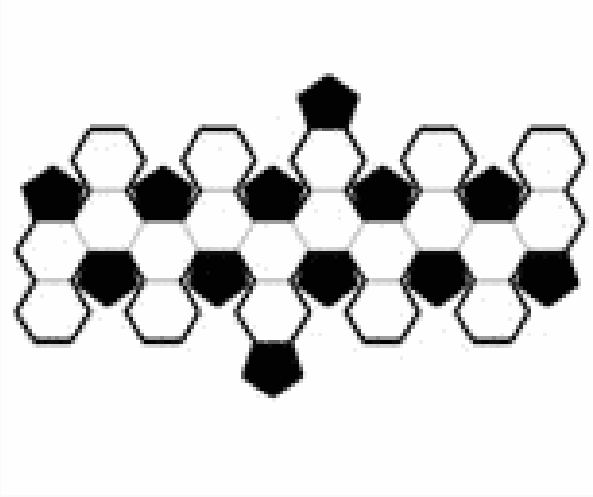
- Kavrayskiy VII

UV-Mapping



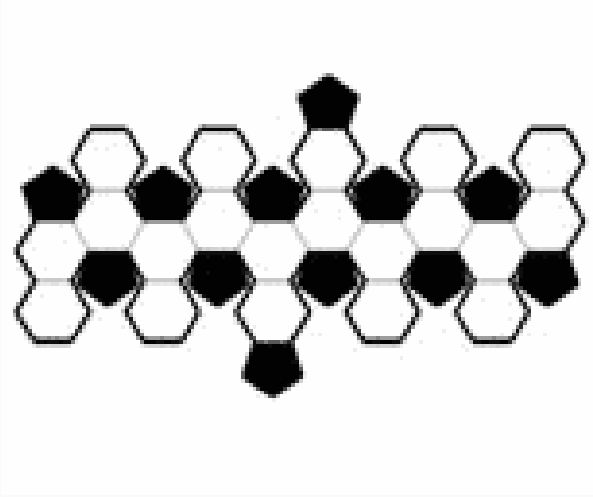
- In graphics known as UV-mapping
- Usually we get this from a modelling program such as Maya
- Unfold mesh to a plane

UV-Mapping



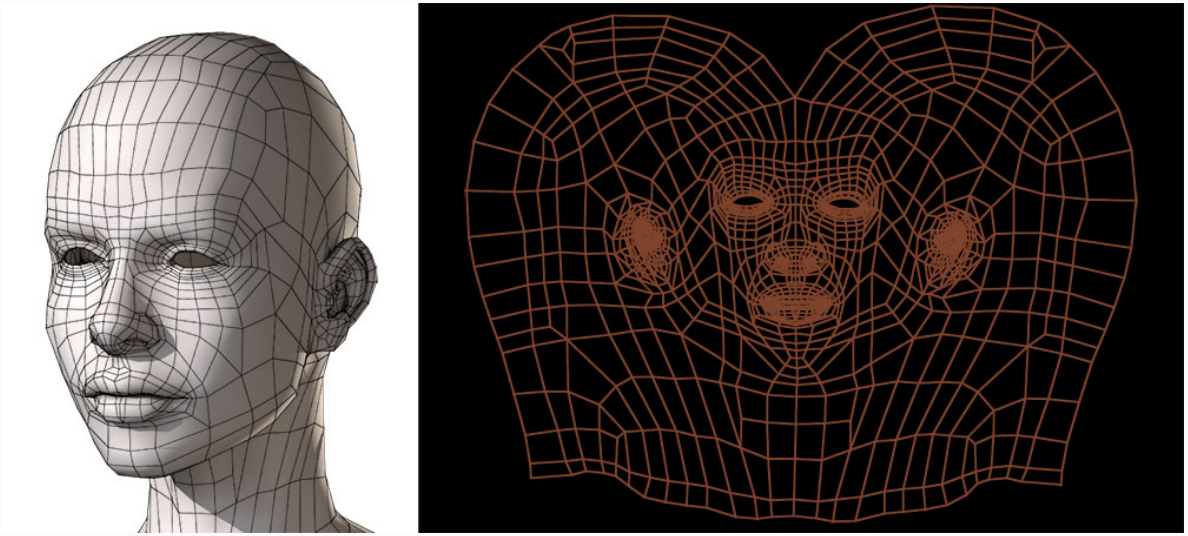
- In graphics known as UV-mapping
- Usually we get this from a modelling program such as Maya
- Unfold mesh to a plane
- *Can you unfold any triangle mesh to a plane without overlap?*

UV-Mapping



- In graphics known as UV-mapping
- Usually we get this from a modelling program such as Maya
- Unfold mesh to a plane
- *Can you unfold any triangle mesh to a plane without overlap?*
- Considered an unsolved problem

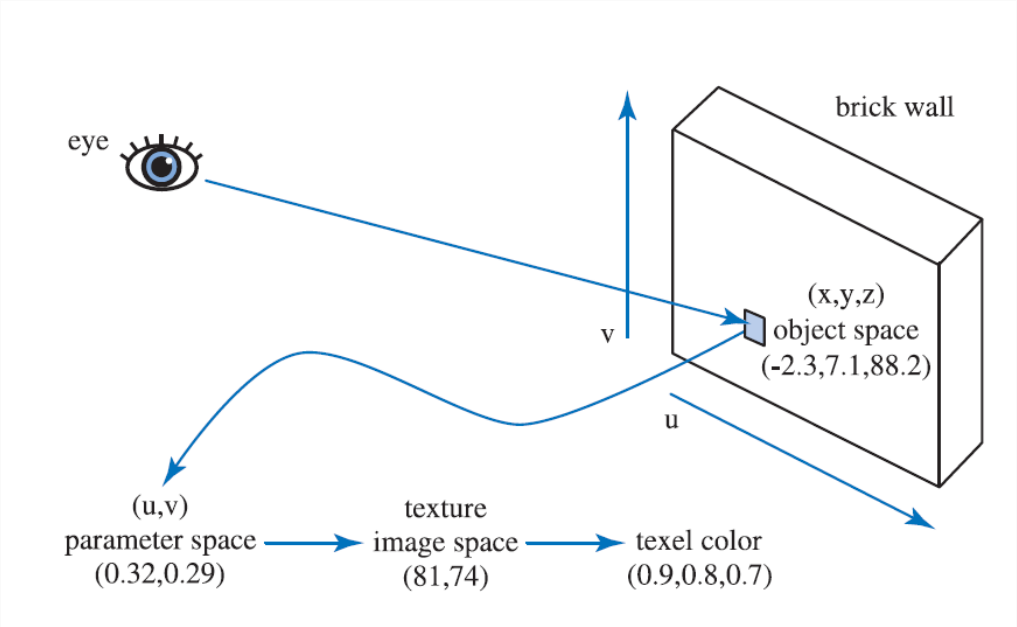
Texture mapper[2]



- Lets assume we have a triangle mesh with associated texture coordinates
- lets interpolate textures. . .
- Remember that in hardware we have Gouraud normally which means only the interpolated quantites can be affected by the texture, for example not normals

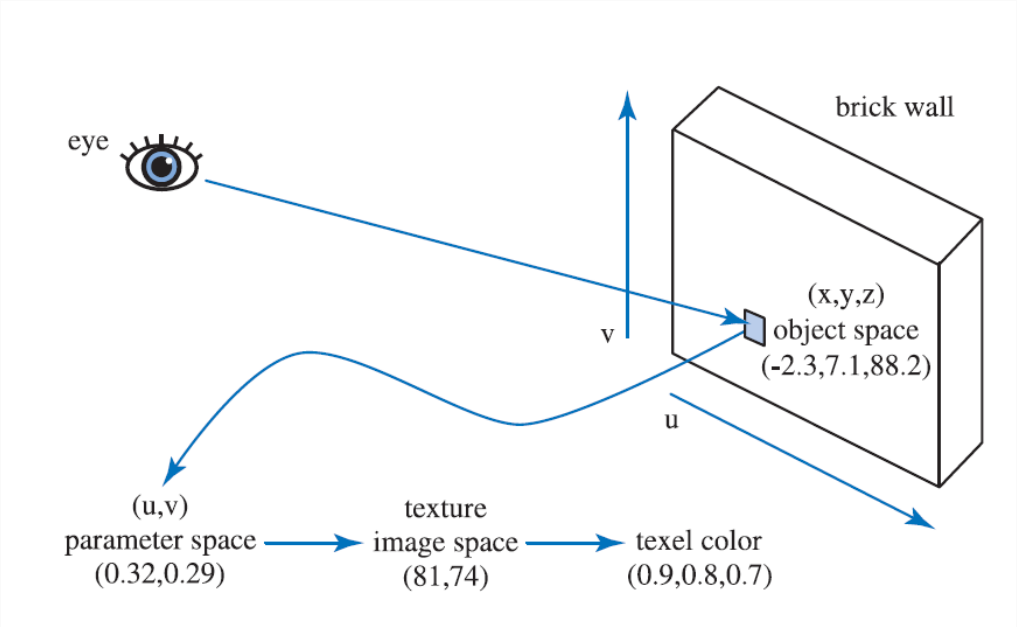
[2] URL

Texture Mapping



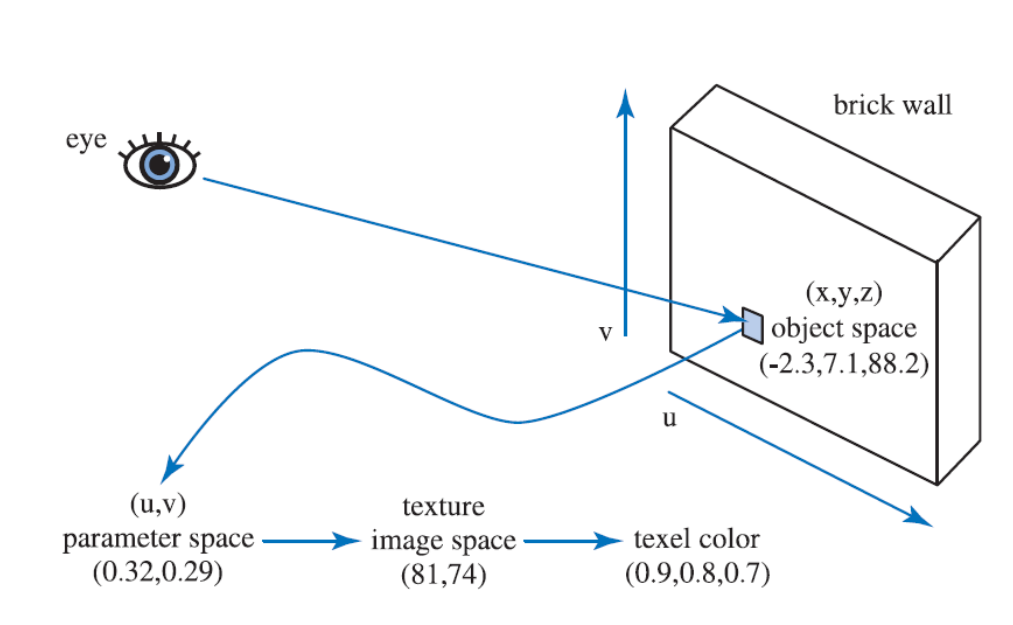
1. Compute (x, y, z) coordinate in world

Texture Mapping



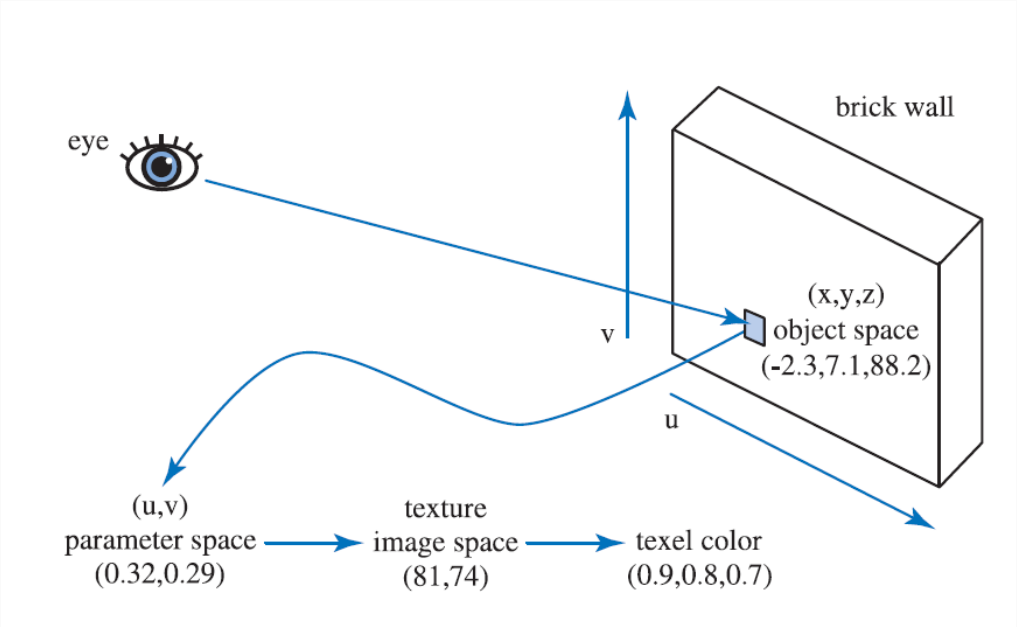
1. Compute (x, y, z) coordinate in world
2. Map to polygon (u, v)

Texture Mapping



1. Compute (x, y, z) coordinate in world
2. Map to polygon (u, v)
3. Map continous (u, v) to discrete coordinates $(u, v)_{map}$

Texture Mapping



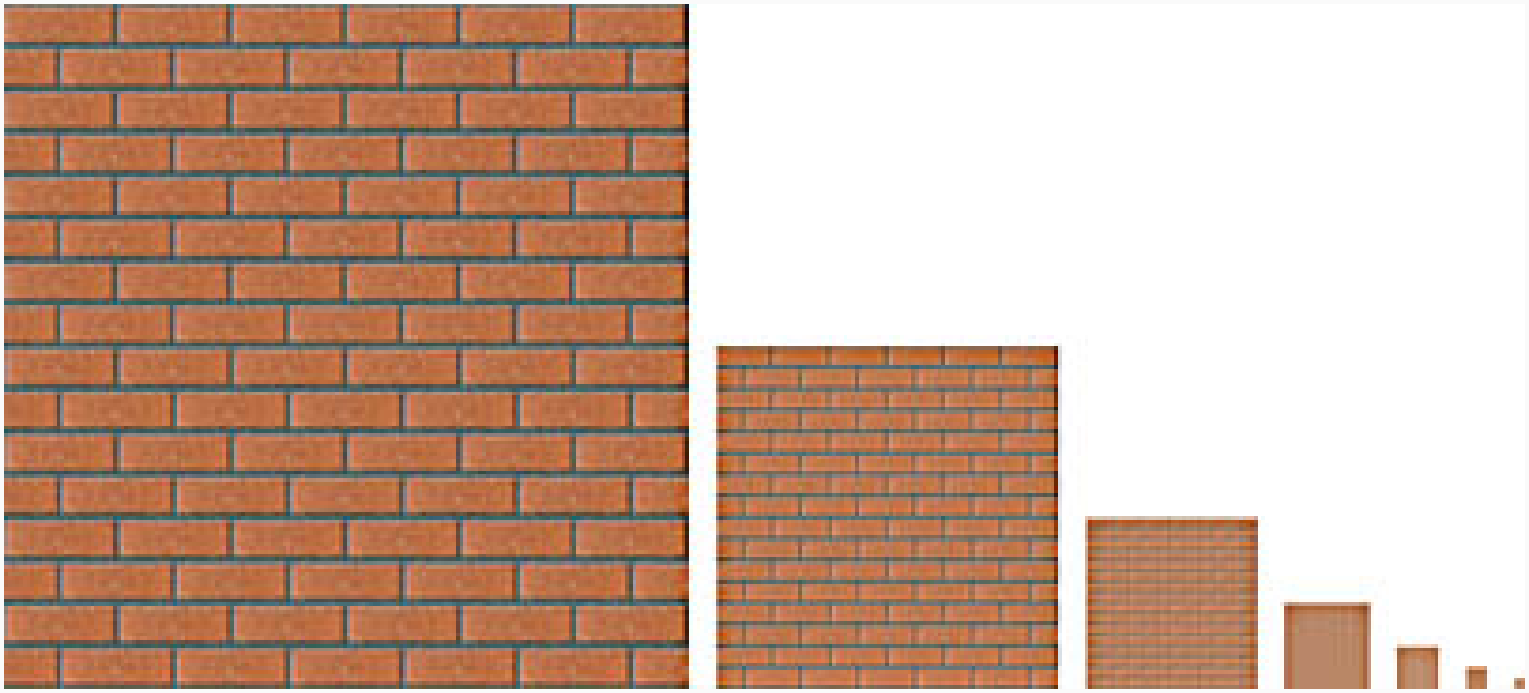
1. Compute (x, y, z) coordinate in world
2. Map to polygon (u, v)
3. Map continous (u, v) to discrete coordinates $(u, v)_{map}$
4. Look up "texture" value (color, normal, etc.)

Texture Filtering

nearest nb. bilinear trilinear

- Space is continous but textures are discrete
- Truncation
  - Zoom In blocky textures
  - Zoom Out aliasing
- Interpolate to get smoother results

Mipmapping



- Pre-compute texture pyramid
- Compute where continous values of polygon lies
- interpolate both in depth and space $\rightarrow$ tri-linear interpolation

Mipmapping

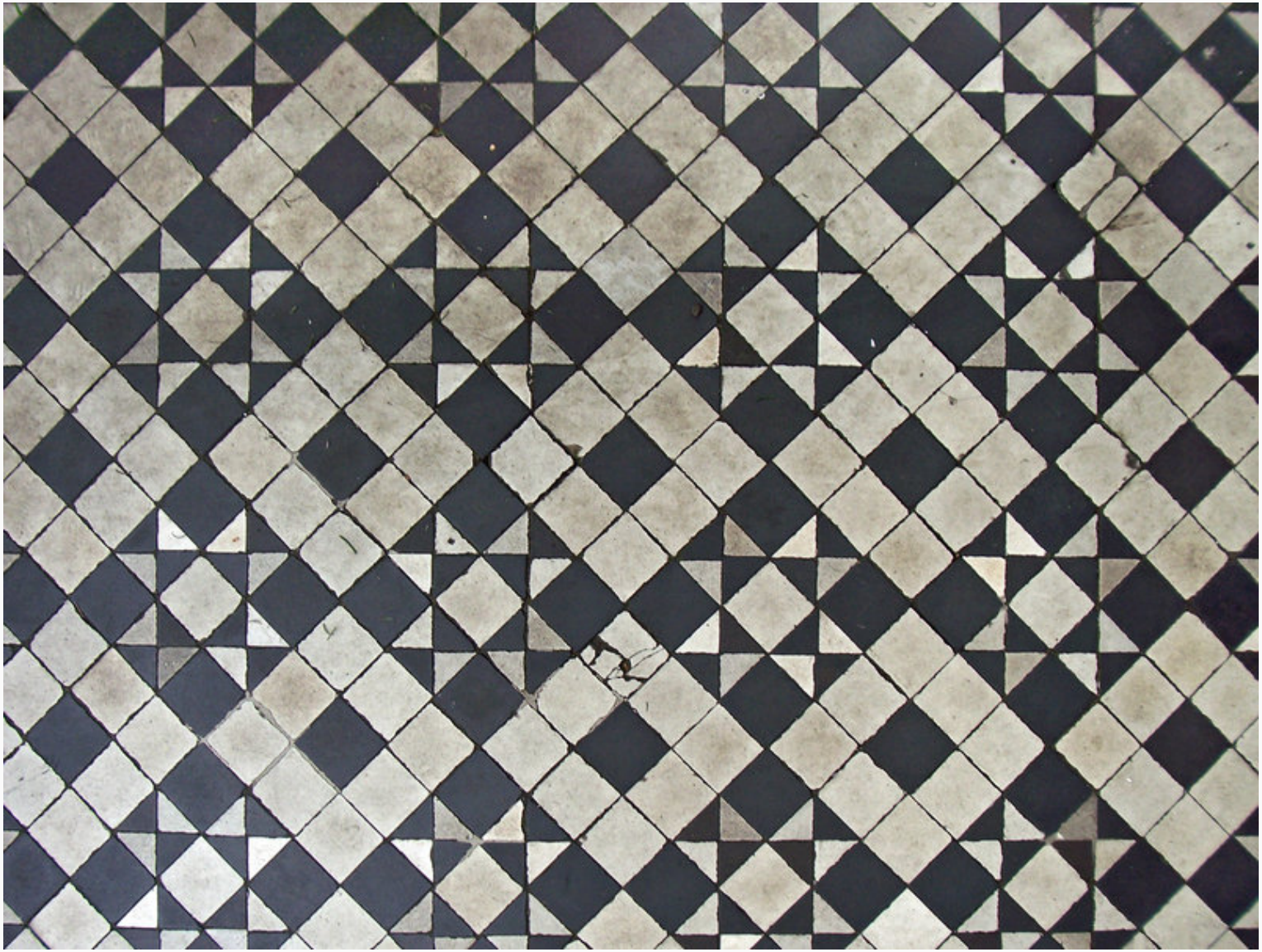
Texture Storage

- Textures takes a lot of space
- 256×256 texture
  - Mipmapps $256 \times 256 + 128 \times 128 + 64 \times 64 + 32 \times 32 = 255$ kb
- Transfer between GPU and CPU memory leads to context switch (expensive)
- Generate or repeat/tile texture

Texture Storage

- simple things
  1. load only parts of mipmap that is needed
  2. group polygons with similar texture together to avoid cache hits
  3. combine texture into sizes that fits cache
- process the texture in cache friendly order
- Loading of textures
  - pre-fetch
  - time-stamp textures

Tiling Textures



Compositing

- Vertex shader
  - Computes "values" at vertex and then interpolates this value across the polygon
  - Can run several passes over a polygon and use values that are "already there"
- Pixel shader
  - Computes "values" at each pixel

Vertexshaded "pixleshader"

- Set the triangle vertices to shade of white

Vertexshaded "pixleshader"

- Set the triangle vertices to shade of white
- Gouraud shade this colour across polygon

Vertexshaded "pixleshader"

- Set the triangle vertices to shade of white
- Gouraud shade this colour across polygon
- Texture map with image

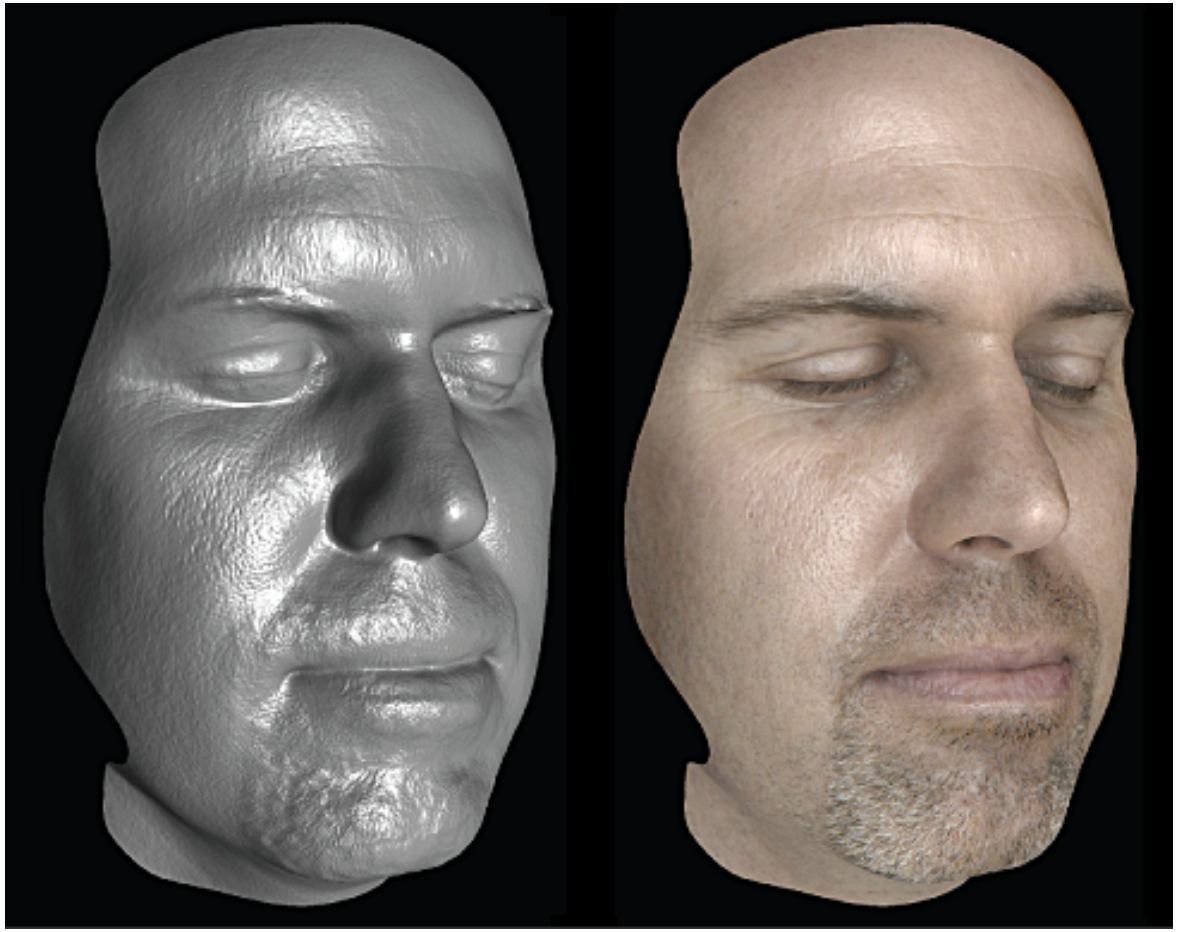
Vertexshaded "pixleshader"

- Set the triangle vertices to shade of white
- Gouraud shade this colour across polygon
- Texture map with image
- modulate texture with white colour

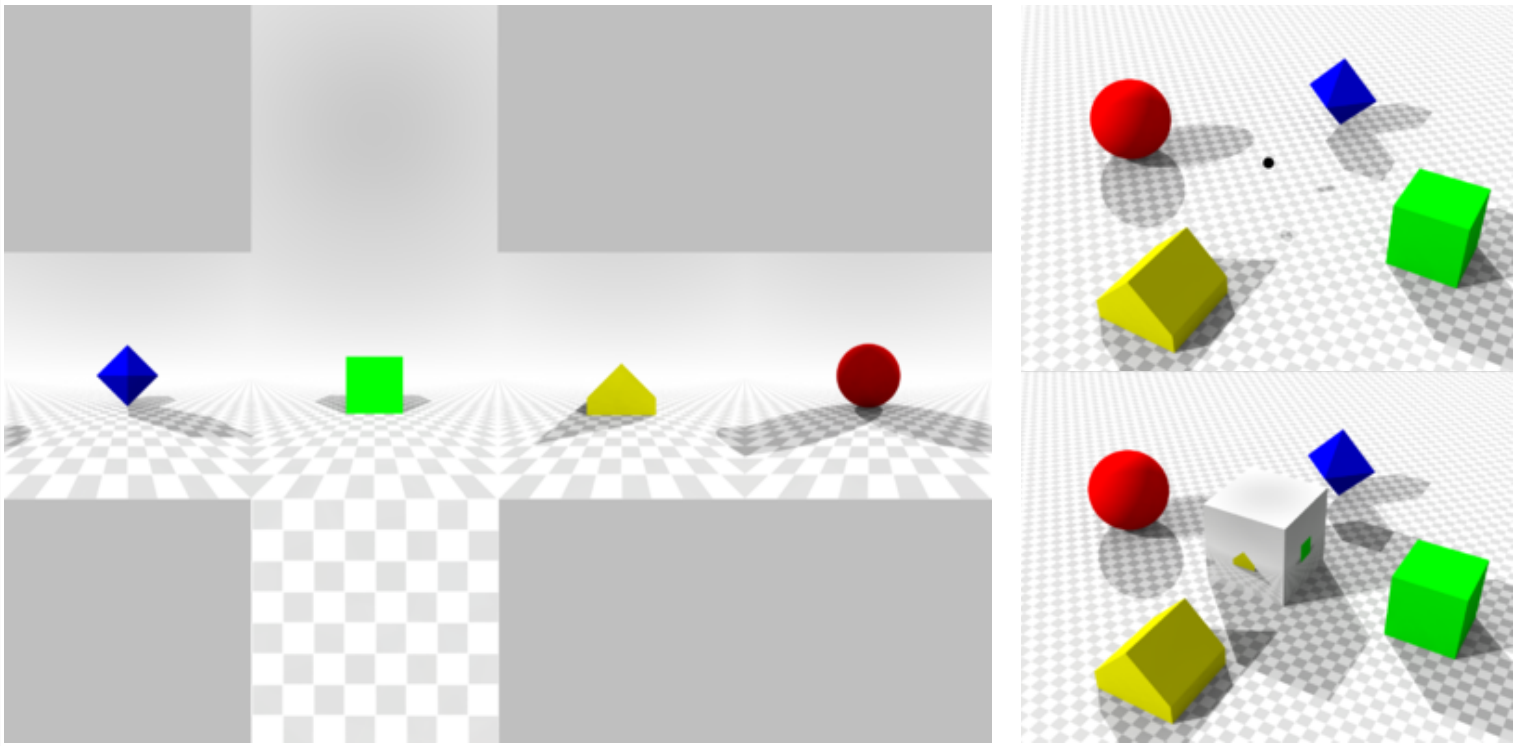
Vertexshaded "pixleshader"

- Set the triangle vertices to shade of white
- Gouraud shade this colour across polygon
- Texture map with image
- modulate texture with white colour
- $\Rightarrow$ lit texture

Specular map

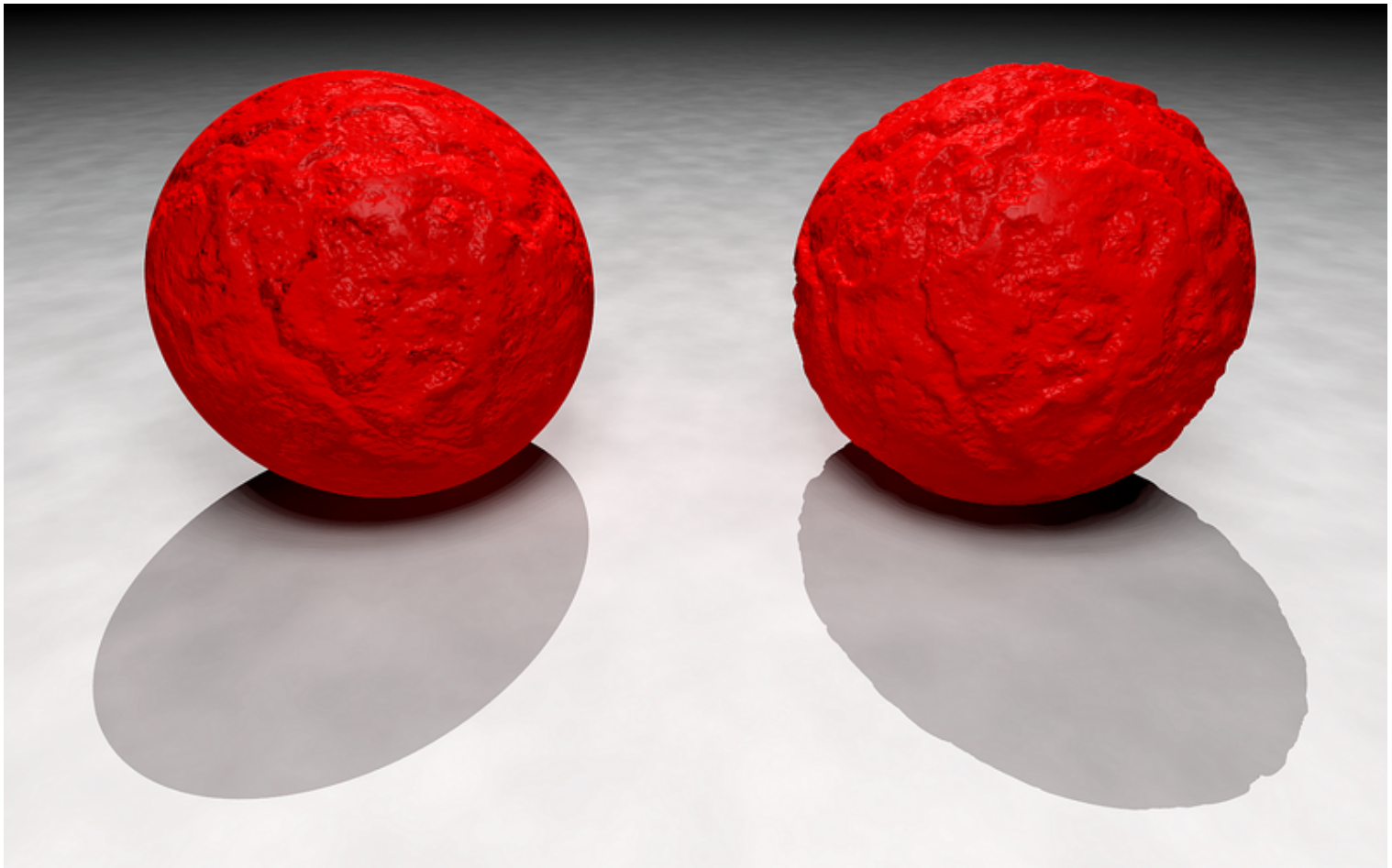


Misc



Misc

Misc



Buffers

Buffers

Frame stores pixels, end product

Depth depth of each pixel

- oclussion culling
- fog
- depth of field, etc.

Stencil discrete buffer per pixel

- mask what to draw in framebuffer

Accumulation acculimate frame information

- combine effects

Stencil Buffer

- defines a mask usually `8-bit` which effects if pixel is rendered
- stencil test (same as depth test)
  - `sfail, dpfail, dppass`
  - `GL_KEEP, GL_ZERO, GL_REPLACE, GL_INCR, GL_INCR_WRAP, GL_DECR, GL_DECR_WRAP, GL_INVERT`
- alter stencil based on test result

Stencil Buffer

Reflections

- draw object frame and stencil buffer
- flip object along `z-axis`
- draw reflection area in stencil buffer
- draw flipped object with stencil test such that on floor but not on object

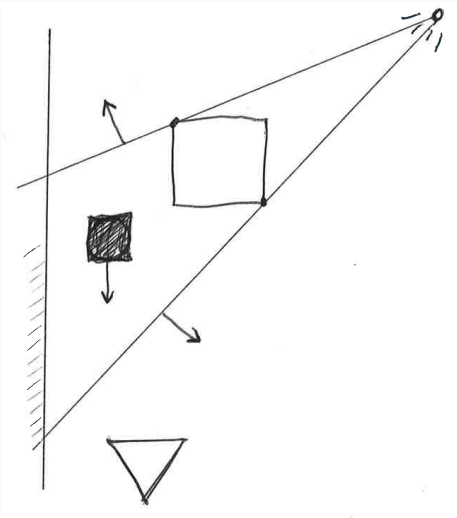
Stencil Shadows[2]

Stencil Shadows

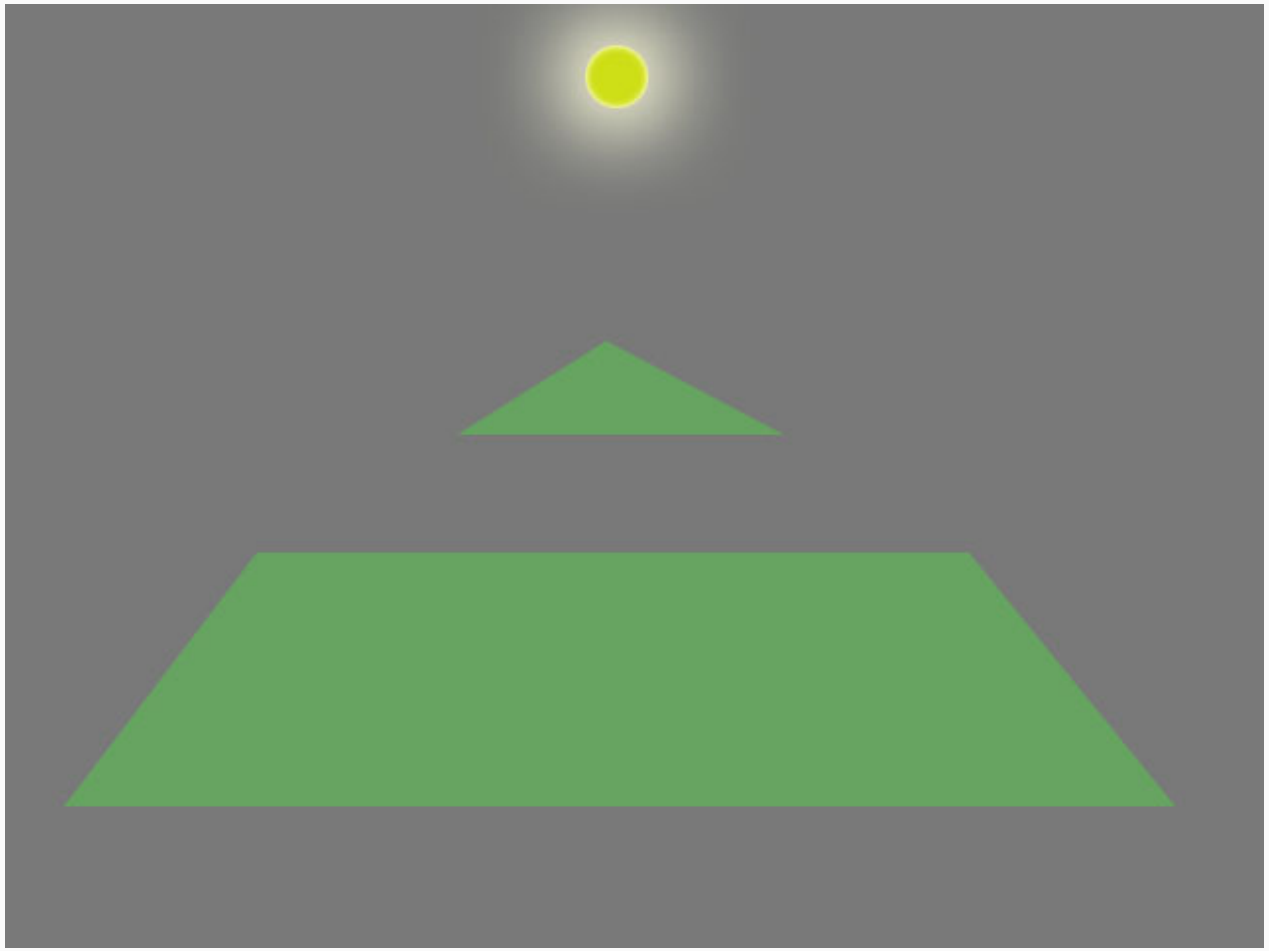
- Draw world in ambient light and update depth buffer
- Compute shadow volume for each object
- Back-face depth-test

  fail increase stencil buffer $+1$
- Front-face depth-text

  fail decrease stencil buffer -1
- Render rest of scene with stencil buffer $= 0$

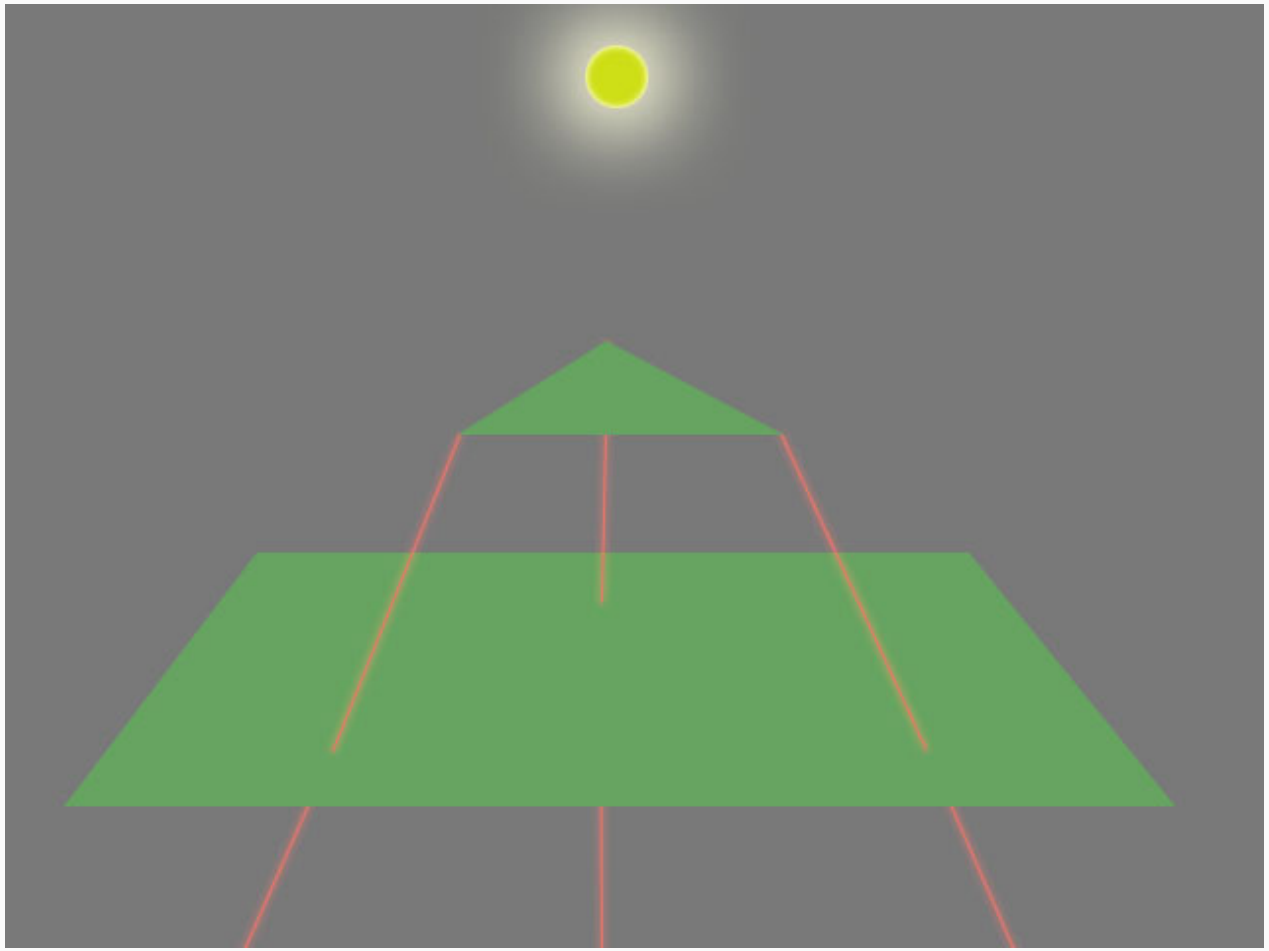


Stencil Shadows[3]



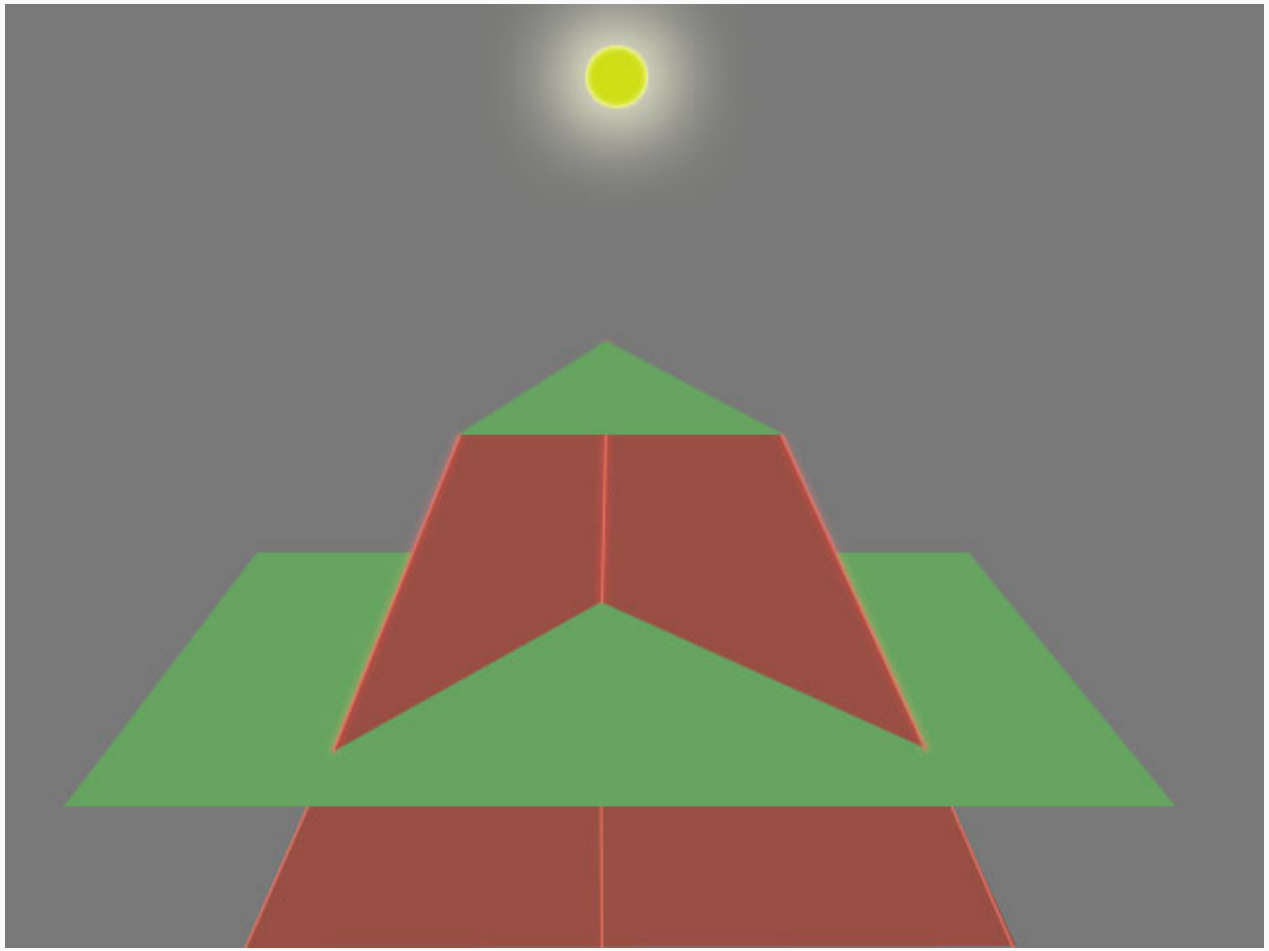
[3] http://www.angelfire.com/games5/duktroa/RealTimeShadowTutorial.htm

Stencil Shadows[3]



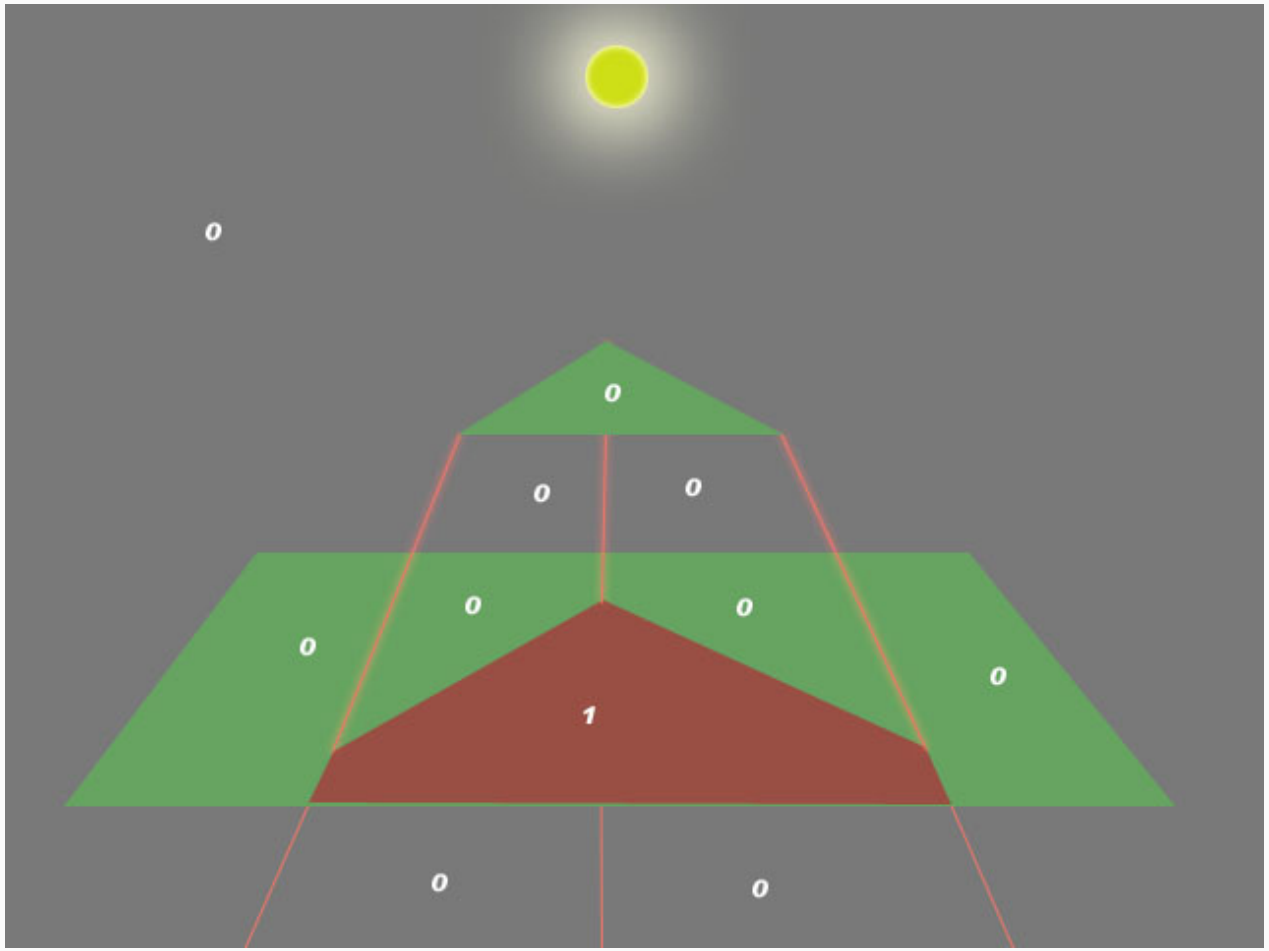
[3] http://www.angelfire.com/games5/duktroa/RealTimeShadowTutorial.htm

Stencil Shadows[3]



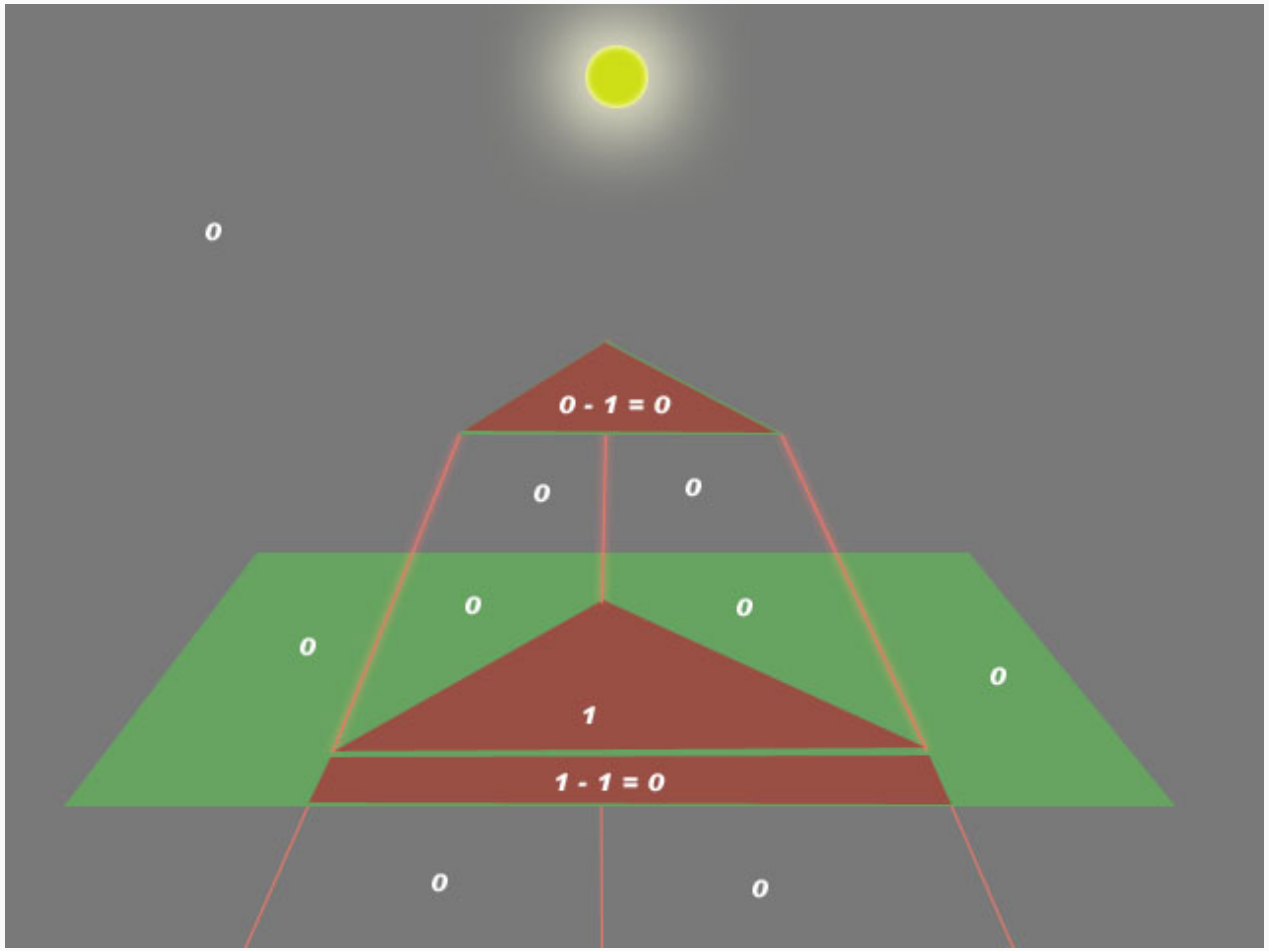
[3] http://www.angelfire.com/games5/duktroa/RealTimeShadowTutorial.htm

Stencil Shadows[3]



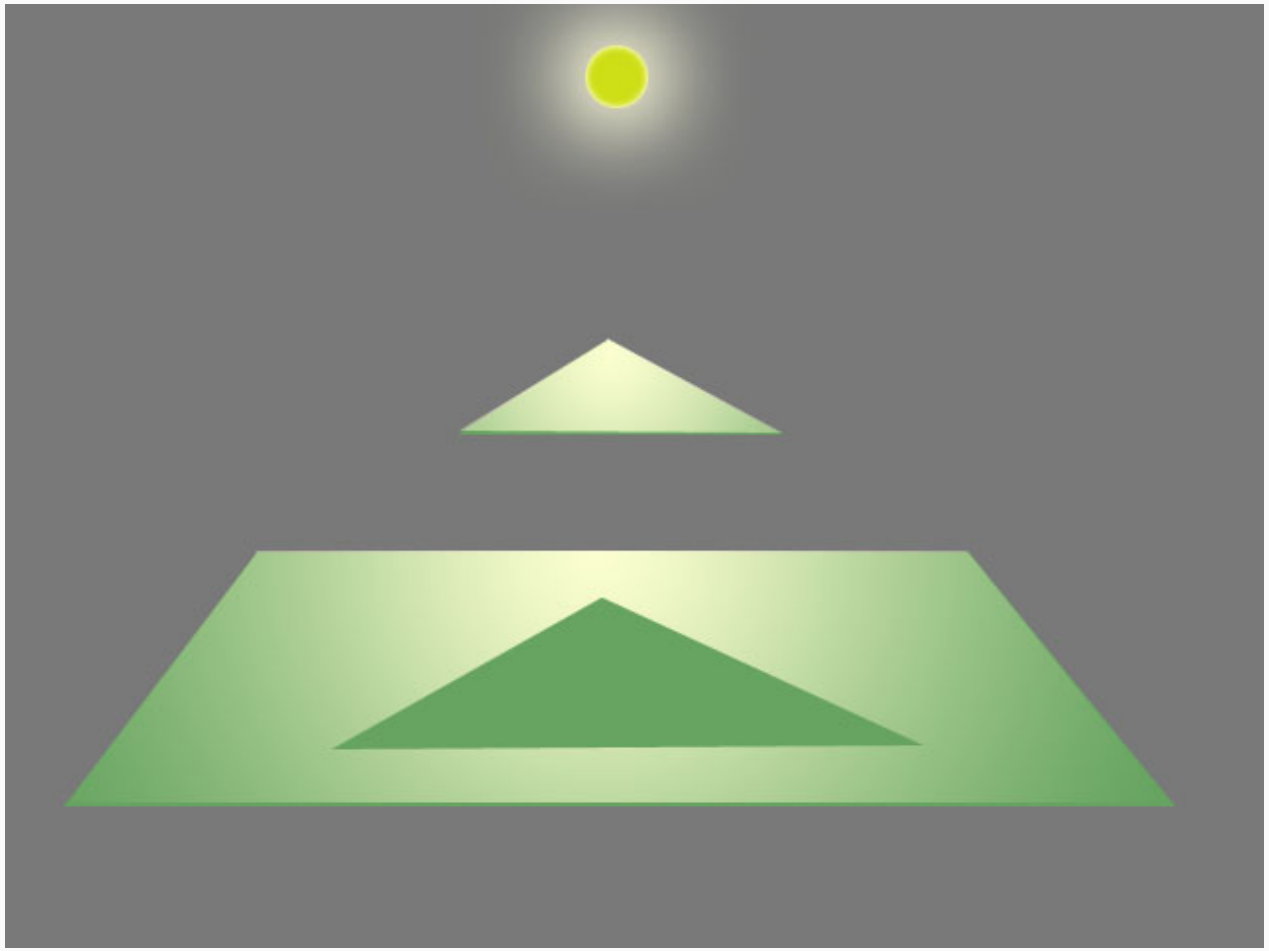
[3] http://www.angelfire.com/games5/duktroa/RealTimeShadowTutorial.htm

Stencil Shadows[3]



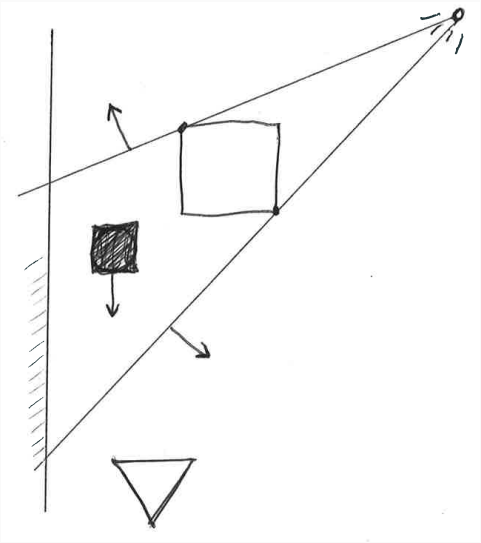
[3] http://www.angelfire.com/games5/duktroa/RealTimeShadowTutorial.htm

Stencil Shadows[3]



[3] http://www.angelfire.com/games5/duktroa/RealTimeShadowTutorial.htm

Shadow Maps

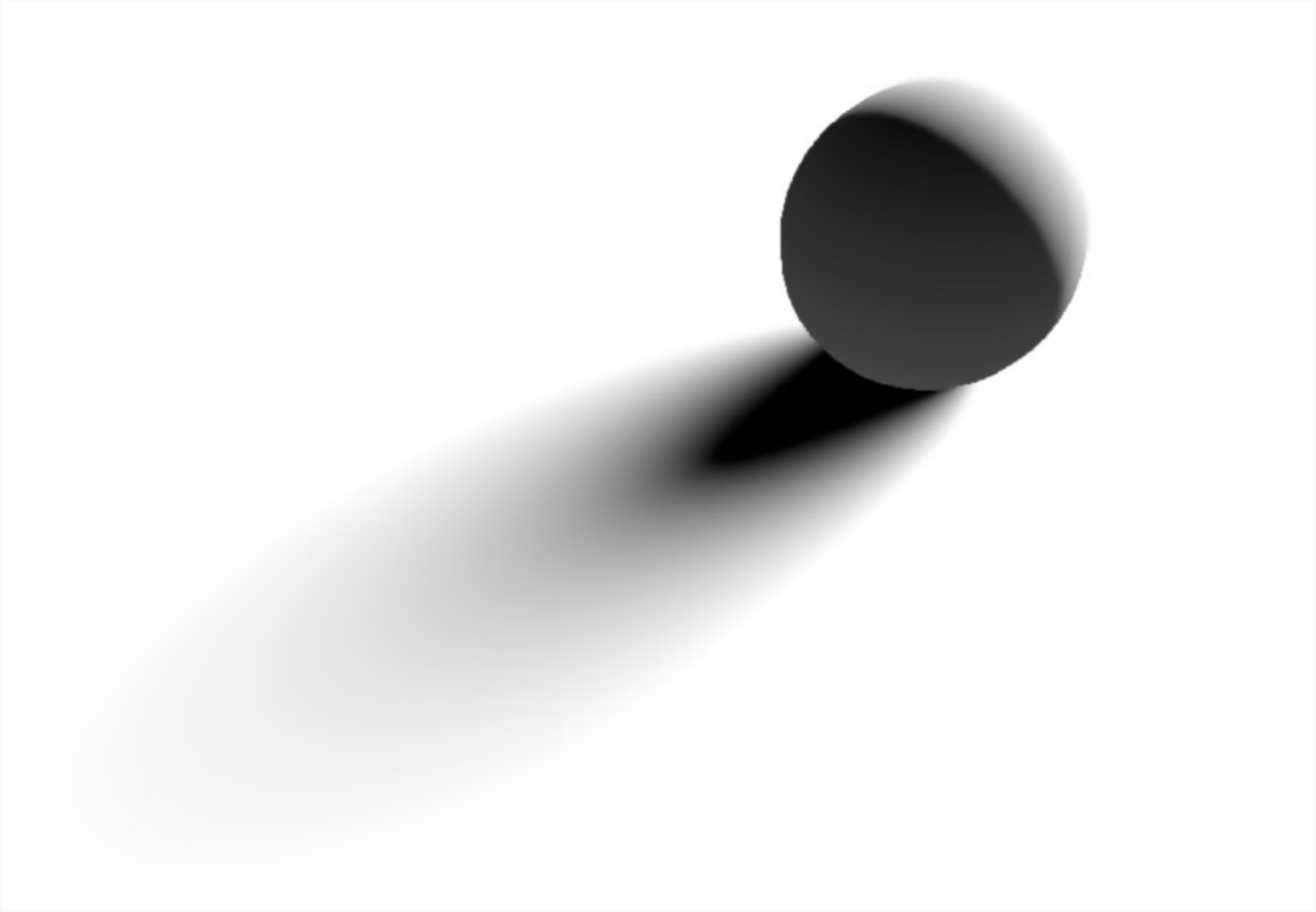


1. Render depth-buffer from light source
2. Render camera view
   - convert $(u, v)_{\text{cam}}$ to $(u, v)_{\text{light}}$ compare z
   - if behind object $\rightarrow$ draw in shadow
   - otherwise draw with light

Accumulation Buffer

- Same size as frame buffer `RGBA`
- Used to do *image processing* on screen
- Always writes every pixel in the buffer
- Very optimised writes (`Blits`) because we are writing a whole chunk of data at the same time
- `OP: ACCUM, LOAD, RETURN, MULT, ADD`

Combinations



Combinations

Combinations

Multi-pass Rendering

Vertex shader Compute information at vertex

- will interpolate this out across triangle
- linear interpolation (Gouraud)
- if you want something else then you would break up geometry (geometry shader)

Pixel shader Compute information at pixel

- take "any" input **AND** interpolated vertex attribute
- sometimes necessary or useful to break up computation in several stages

Run shader in several passes across polygon. We did just that with first Z and then vertex attributes.

ID Software

ID Software

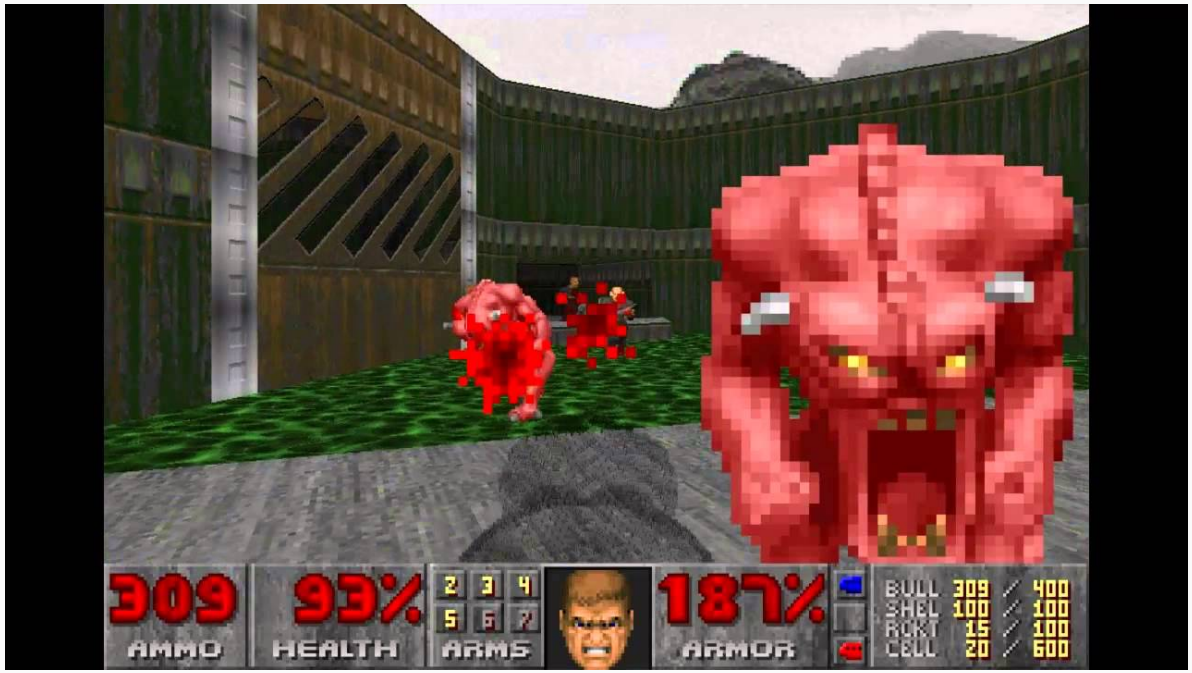
- John Carmack
- ID Software
- “Created” the first person shooter
- Games tells a neat story of how rasterisers developed
- Usually releases engine open source

ID Software

[4] **1992** Raycaster, x,y,height

[4] Wolfenstein 3D

ID Software



[4] **1993** Same as Wolfenstein but added environment maps

[4] ID Tech 1 Doom

ID Software

[4] **1996** Real 3D, Gouraud shading, Both hardware and software rendering

[4] ID Tech 2 Quake

ID Software

[4] **1997** mipmapping

[4] ID Tech 2 Quake

ID Software

2004, Per-pixel lighting, Normal maps [4]

[4] ID Tech 4 Doom 3

ID Software

2011 Textures, very very big textures [4]

[4] ID Tech 5 Rage

ID Software

2016-17 Lots lots of everything ;-) [4]

[4] ID Tech 6 Doom (4)

Summary

Summary

- Interpolation is the key to speed-up rendering
- Vertex shading vs. Pixel shading
- You can run a vertex shader in several passes where the look-up is "on the polygon"
- Buffers can be really useful, especially if you factorise your rendering pipeline

Next Time

Lecture clipping

- how can we make sure that we only draw on the screen
- last bit of rasterisation
- summary of rasterisation

eof

eof