University of BRISTOL

COMS 30115

Shading

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

March 11th, 2019

http://www.carlhenrik.com

Last time

- Lines
- Triangles
  - spantables
  - barycentric coordinates
- Perpective correct interpolation

Today

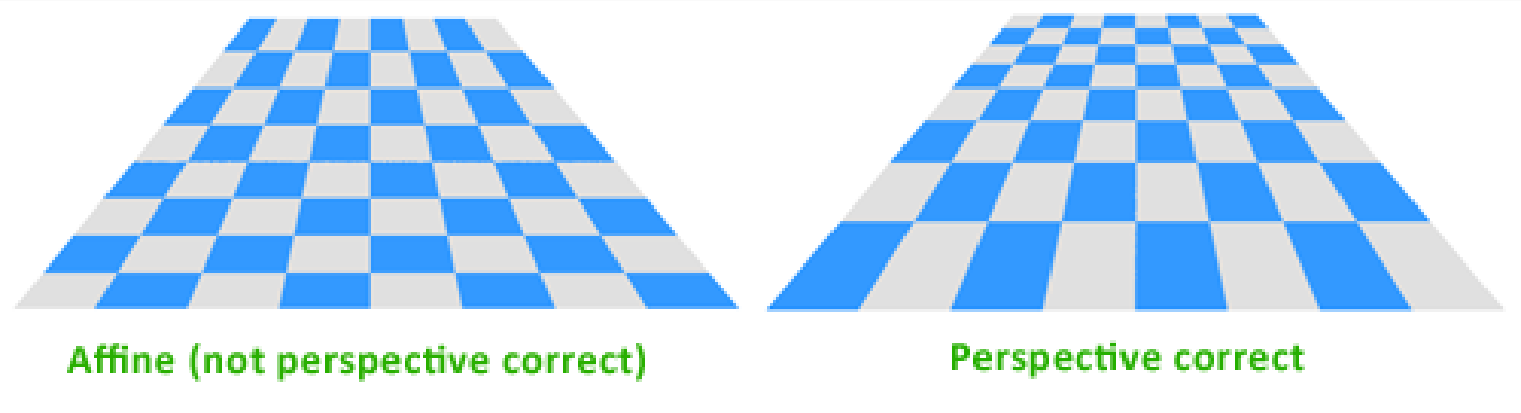
- Perspective correct interpolation of quantities
- Short re-cap on light
- Shading
    - what we want to interpolate

The Book

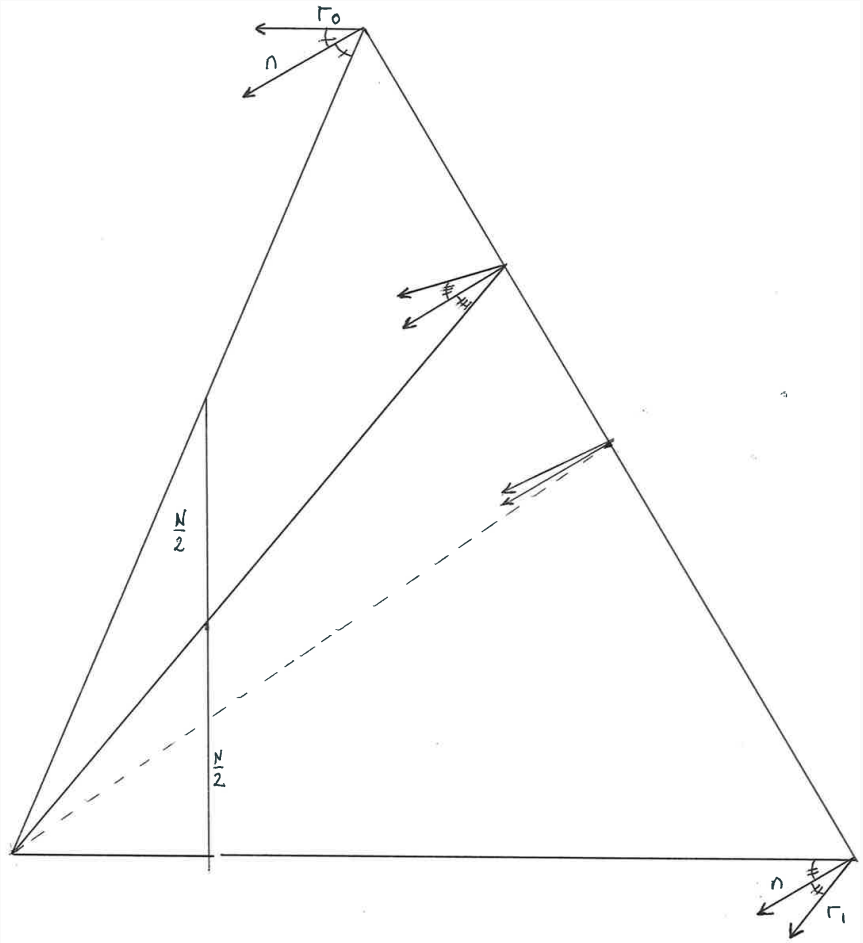
- Perspective Correct Interpolation and Vertex Attributes URL

Perspective Correct

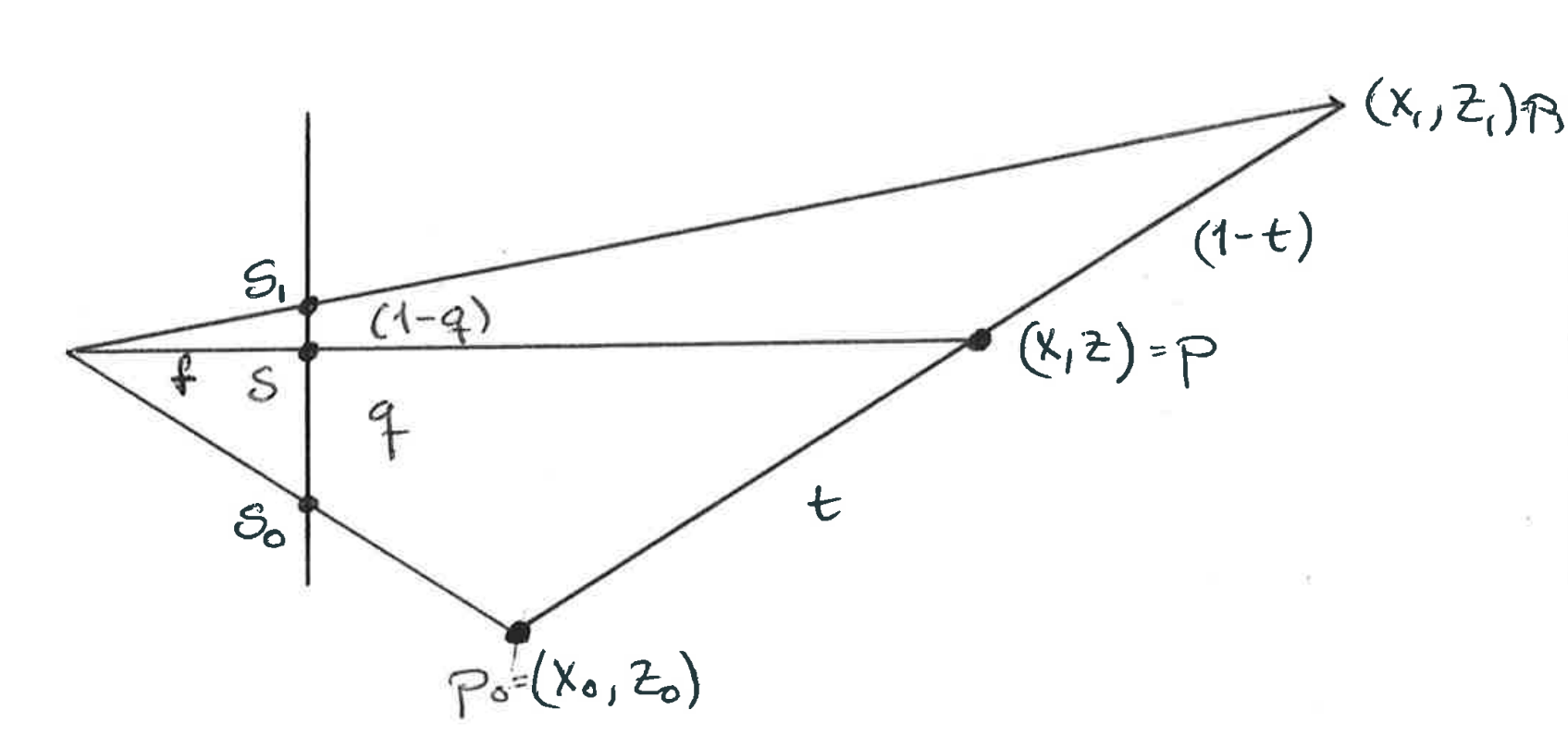
Perspective Correct



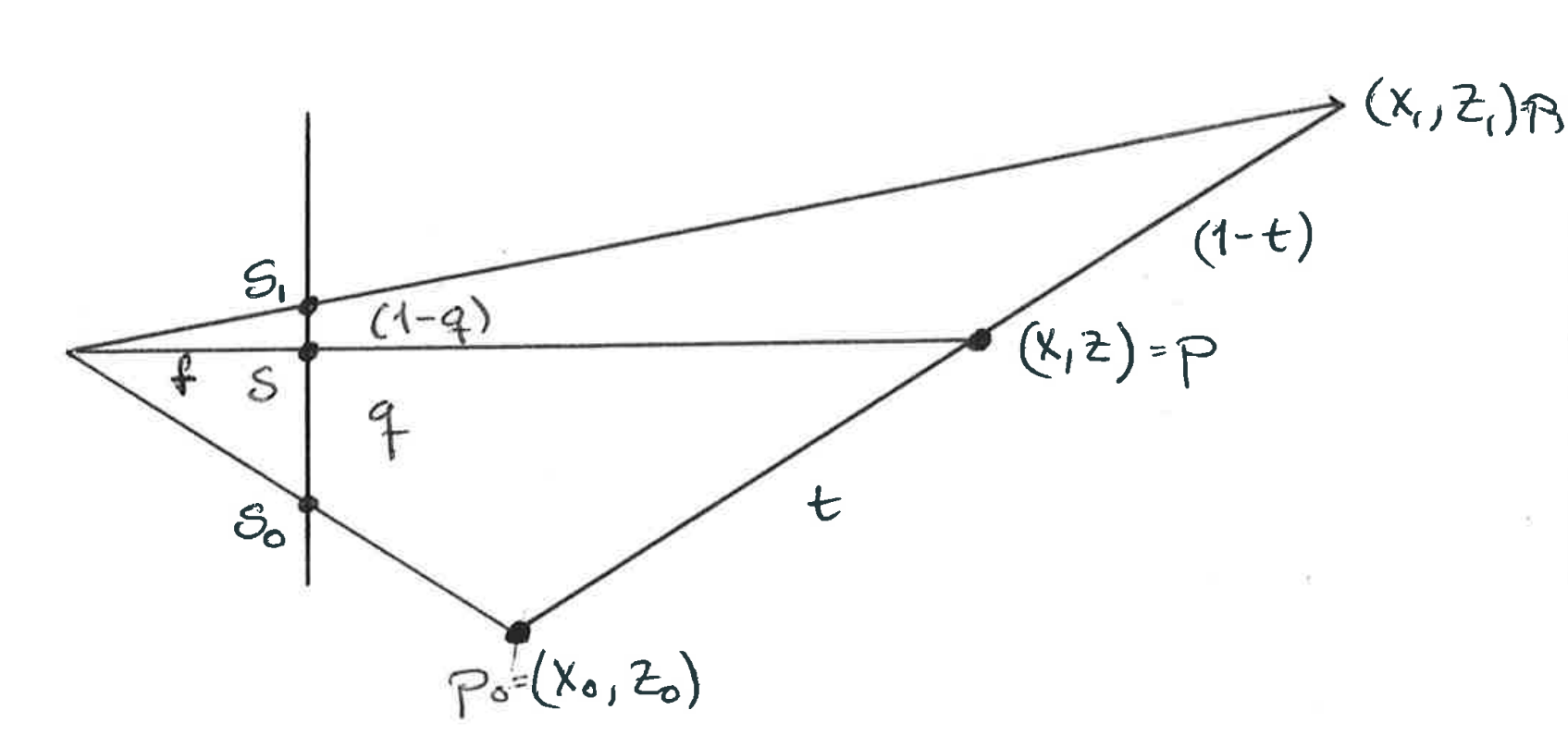
Perspective Correct



Perspective Correct

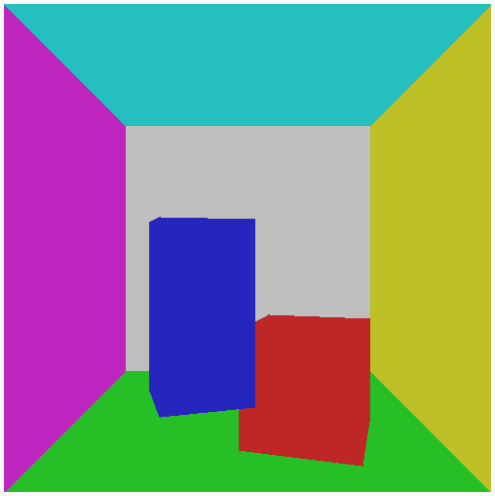


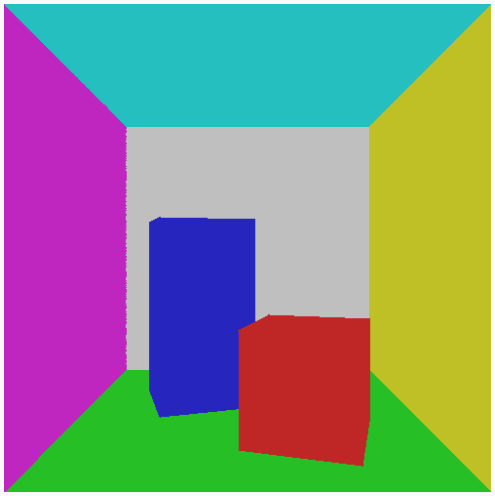
Perspective Correct

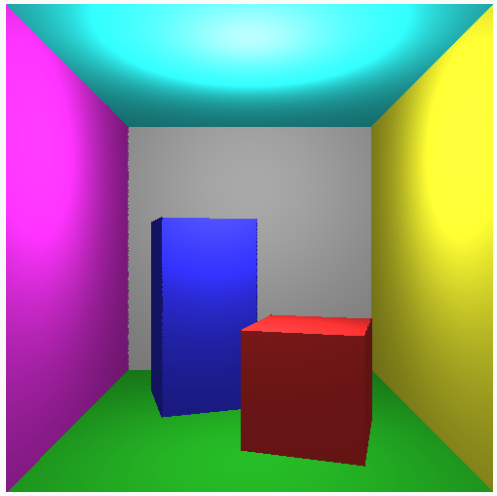


$$\frac{1}{z} = \frac{1}{z_0}(1-q) + \frac{1}{z_1}q$$

Today

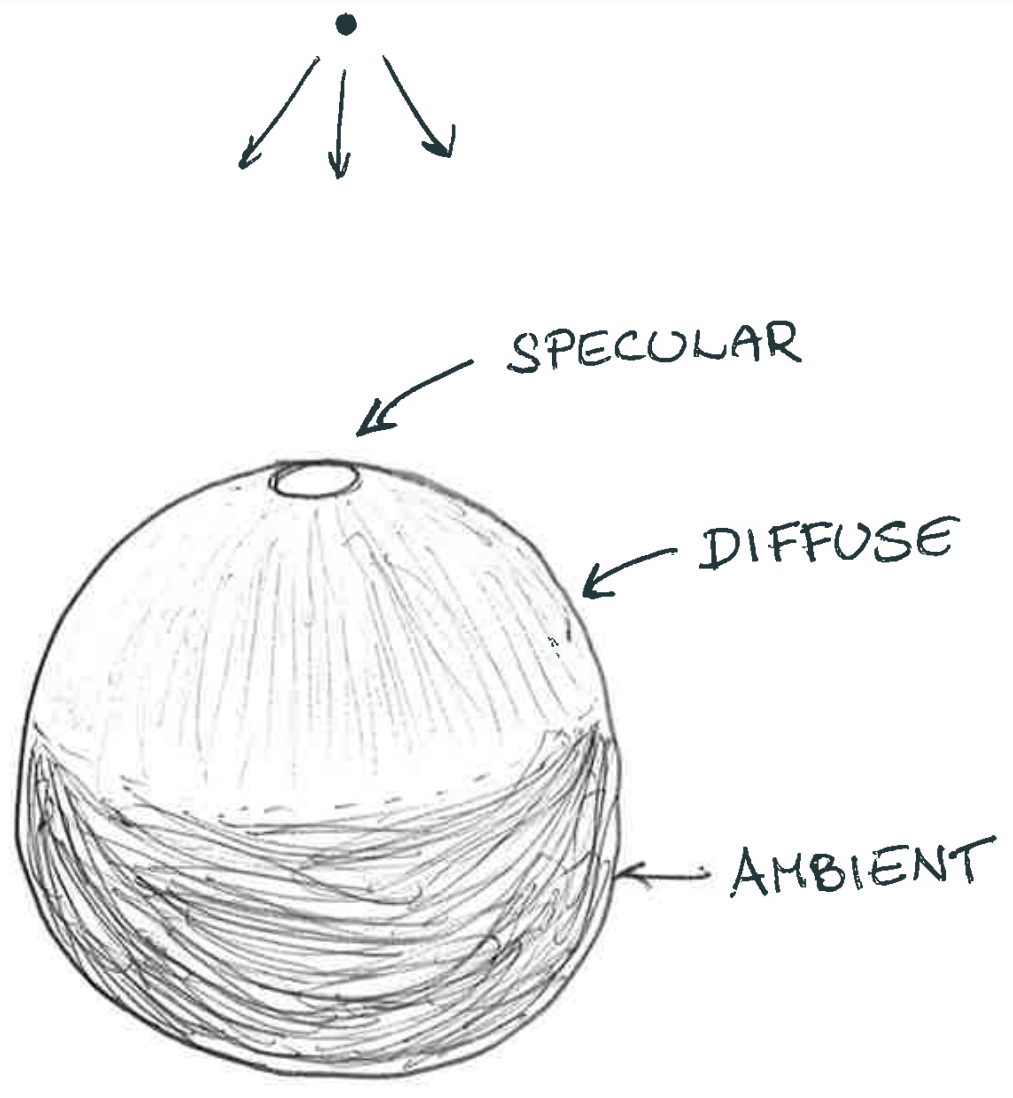






Shading

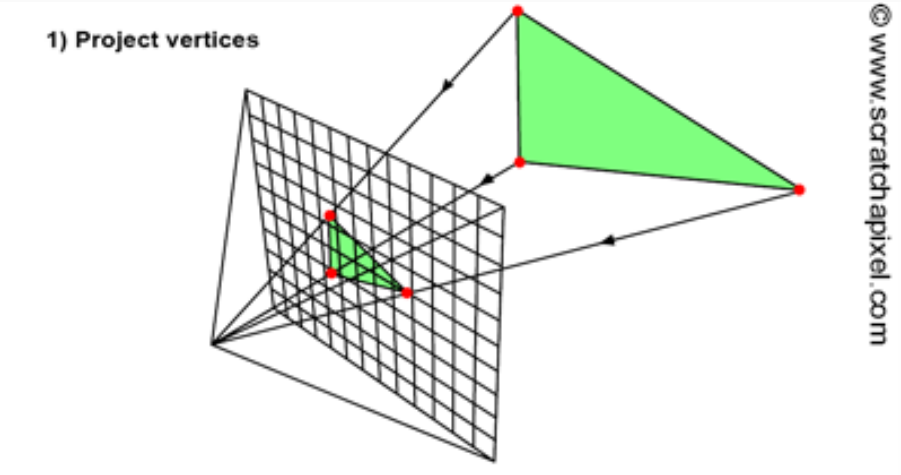
Lights Ball



All Together

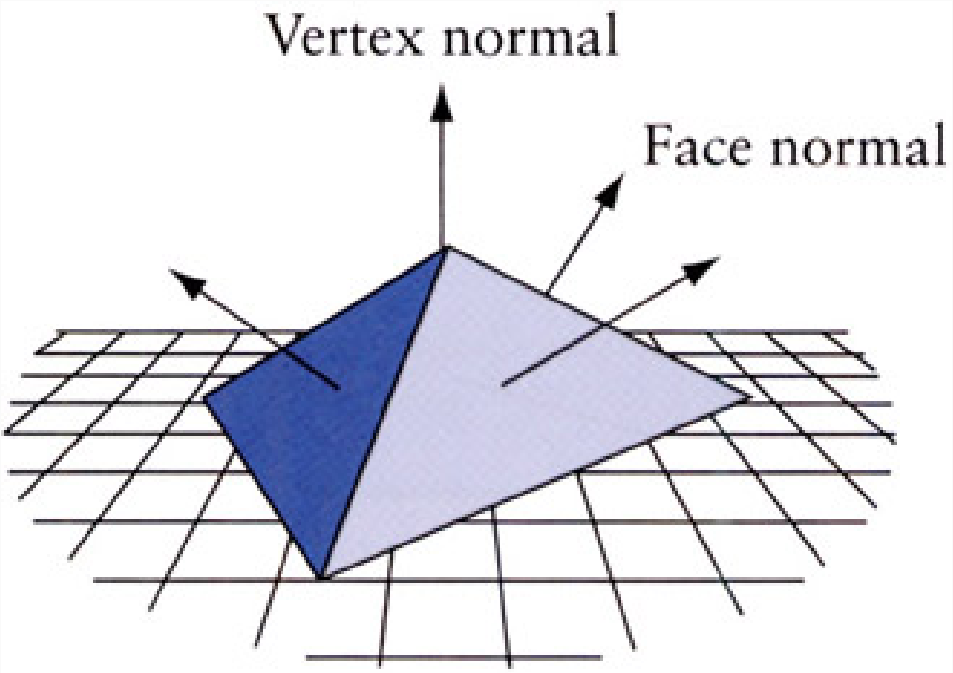
$$
\begin{aligned}
\mathbf{i}_{tot} &= f(\mathbf{i}_{amb}, \mathbf{i}_{diff}, \mathbf{i}_{spec}) \\
&= \mathbf{m}_{emi} + \sum_{i=0}^{N-1} \Big(\mathbf{m}_{amb} \circ \mathbf{s}^i_{amb} \\
&\quad + \frac{\max((\mathbf{n}^{\mathrm{T}}\mathbf{l}^i), 0)\mathbf{m}_{diff} \circ \mathbf{s}^i_{diff} + \max(((\mathbf{r}^i)^{\mathrm{T}}\mathbf{v}), 0)^{m_{shi}} \mathbf{m}_{spec} \circ \mathbf{s}^i_{spec}}{s^i_c + s^i_l \left((\mathbf{s}_{pos} - \mathbf{p})^{\mathrm{T}}(\mathbf{s}_{pos} - \mathbf{p})\right)^{\frac{1}{2}} + s^i_q \left((\mathbf{s}^i_{pos} - \mathbf{p})^{\mathrm{T}}(\mathbf{s}^i_{pos} - \mathbf{p})\right)^{\frac{1}{2}}} \Big)
\end{aligned}
$$

Vertex Shading

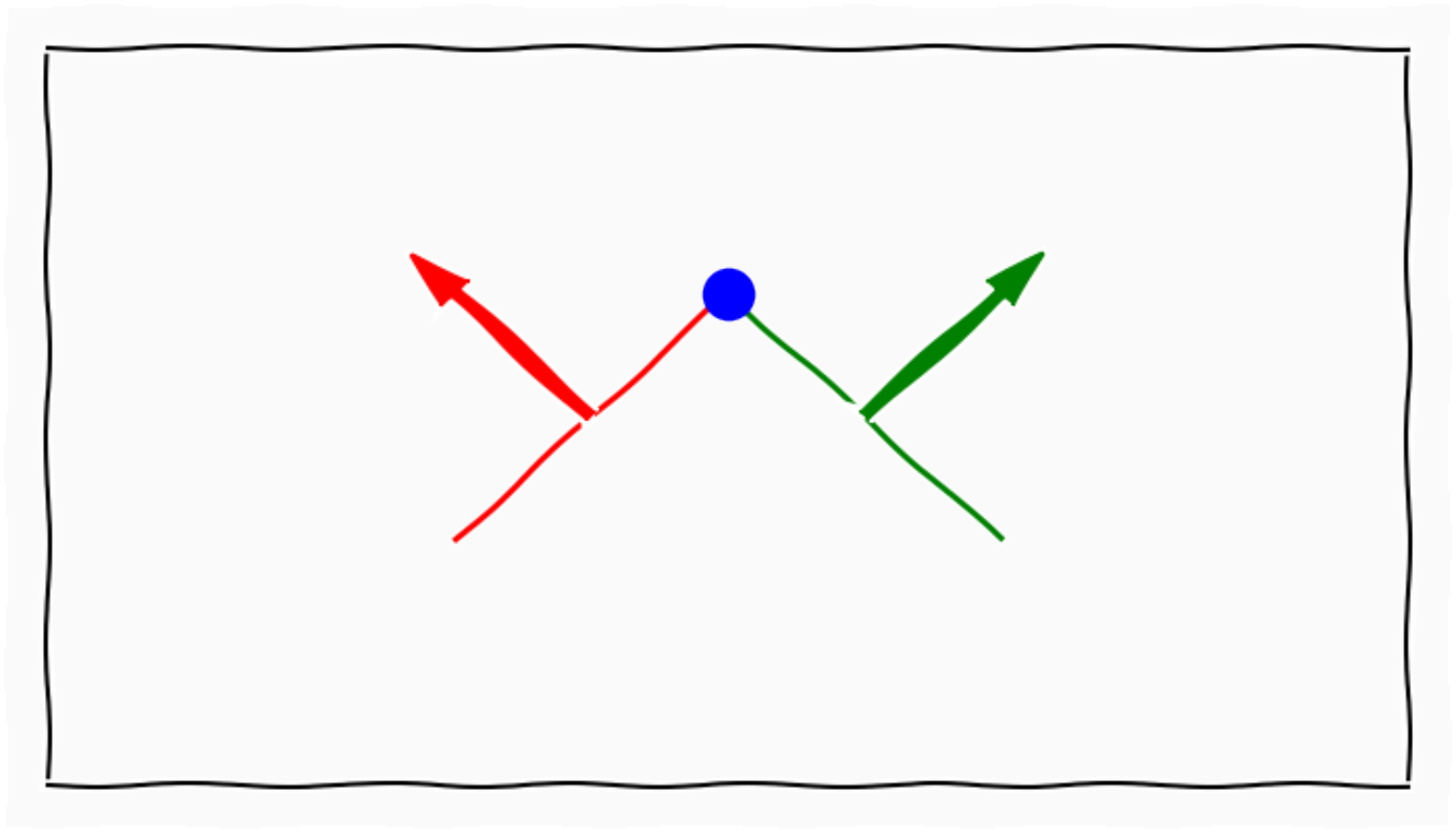


- In the raytracer we did this computation per/pixel
- In the rasteriser we want to do this per/vertex and then interpolate the light
    - make few expensive computations

Normals



Vertex Normals



Flat Shading

- shade for one point (center) of polygon and fill with that colour

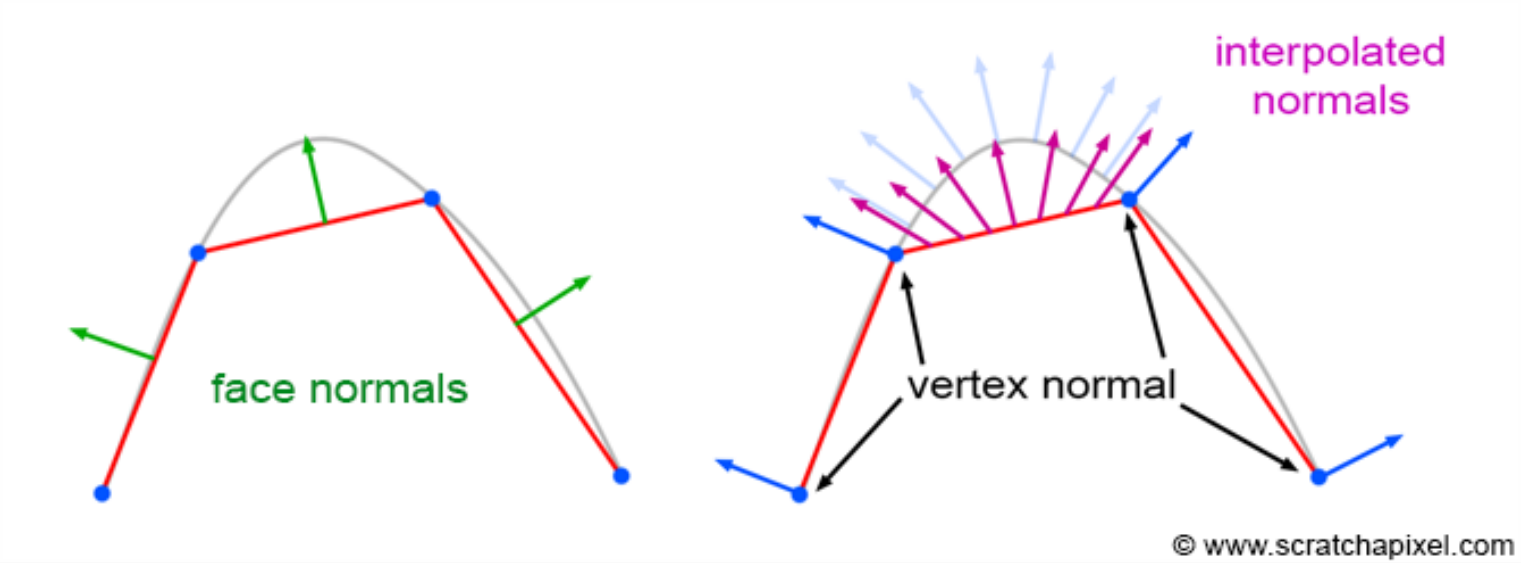
Gouraud Shading

- Compute shading at each vertex

$$i_{a,b,c} = \max(0, \mathbf{l}_{a,b,c}^{\mathrm{T}} \mathbf{n}_{a,b,c})$$

- Interpolate shading across polygon
- Hard to get specular highlights correct

Interpolated Normals



Pixel Shader

- Interpolate vertex normals across polygon

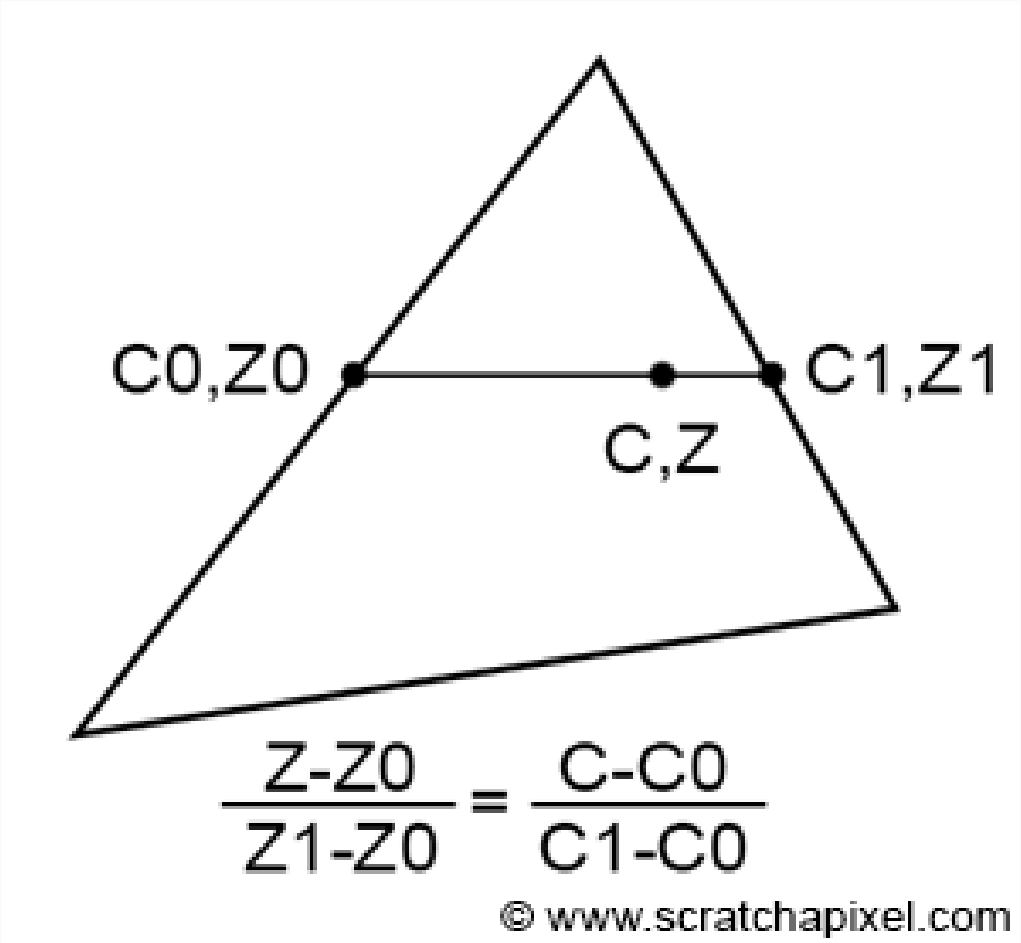
$$\mathbf{n}(u, v) = f(u, v, \mathbf{n}_{left}, \mathbf{n}_{right})$$

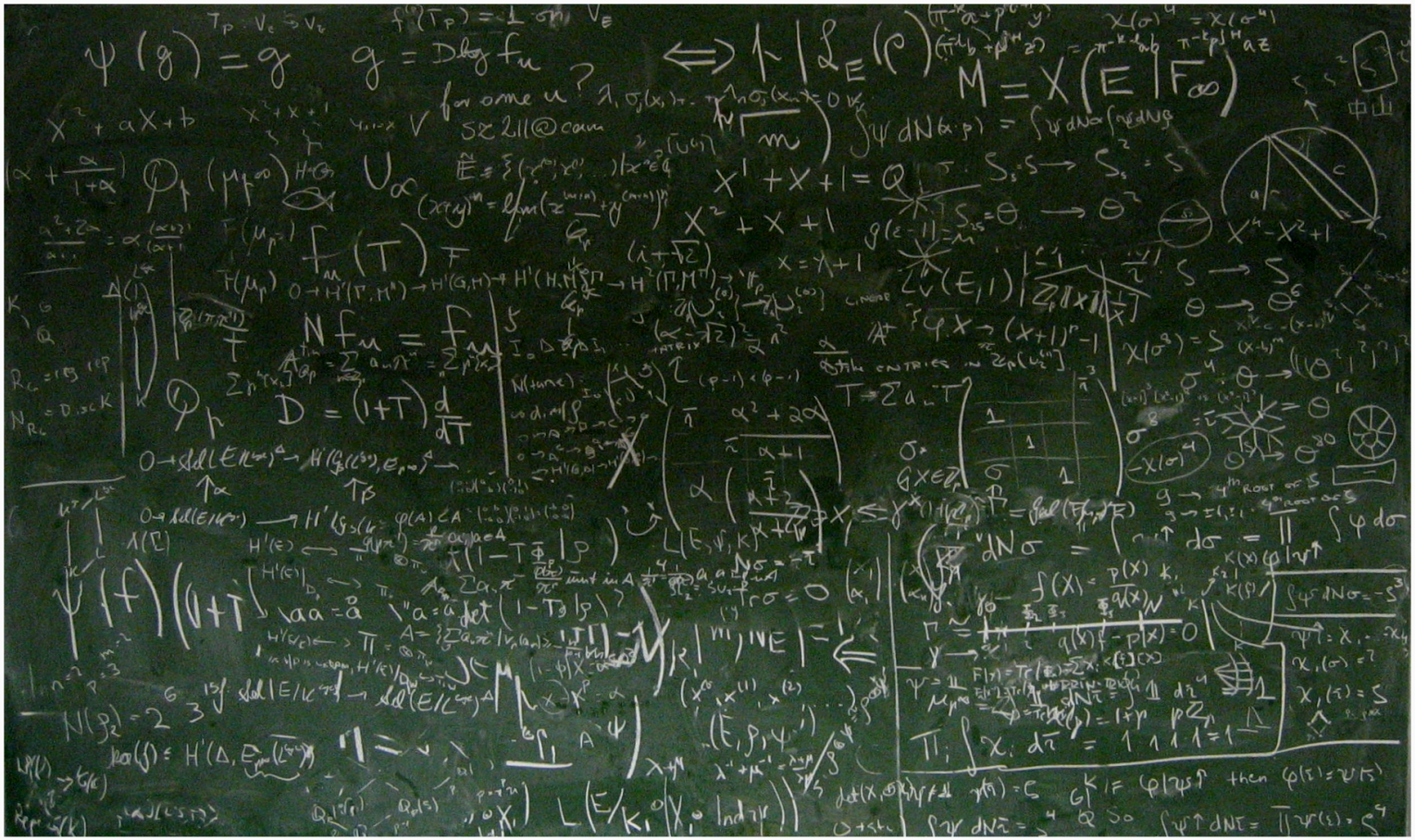
- Compute shading for each pixel

$$i(u, v) = \max(0, \mathbf{l}(u, v)^{\mathrm{T}} \mathbf{n}(u, v))$$

Interpolation of Attributes

Interpolation of Attributes





Perspective Correct Interpolation

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

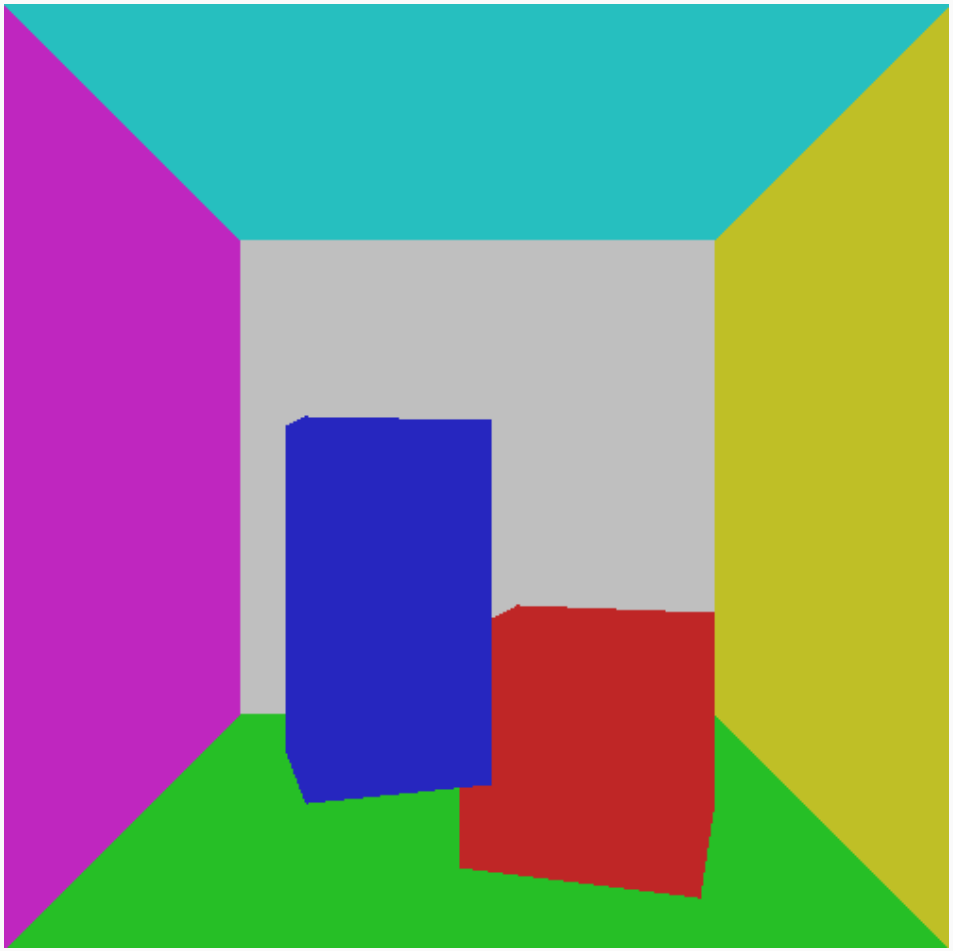
- Most likely the most important equations in a rasteriser
- Write a really flexible function that can interpolate perspective correct

Code

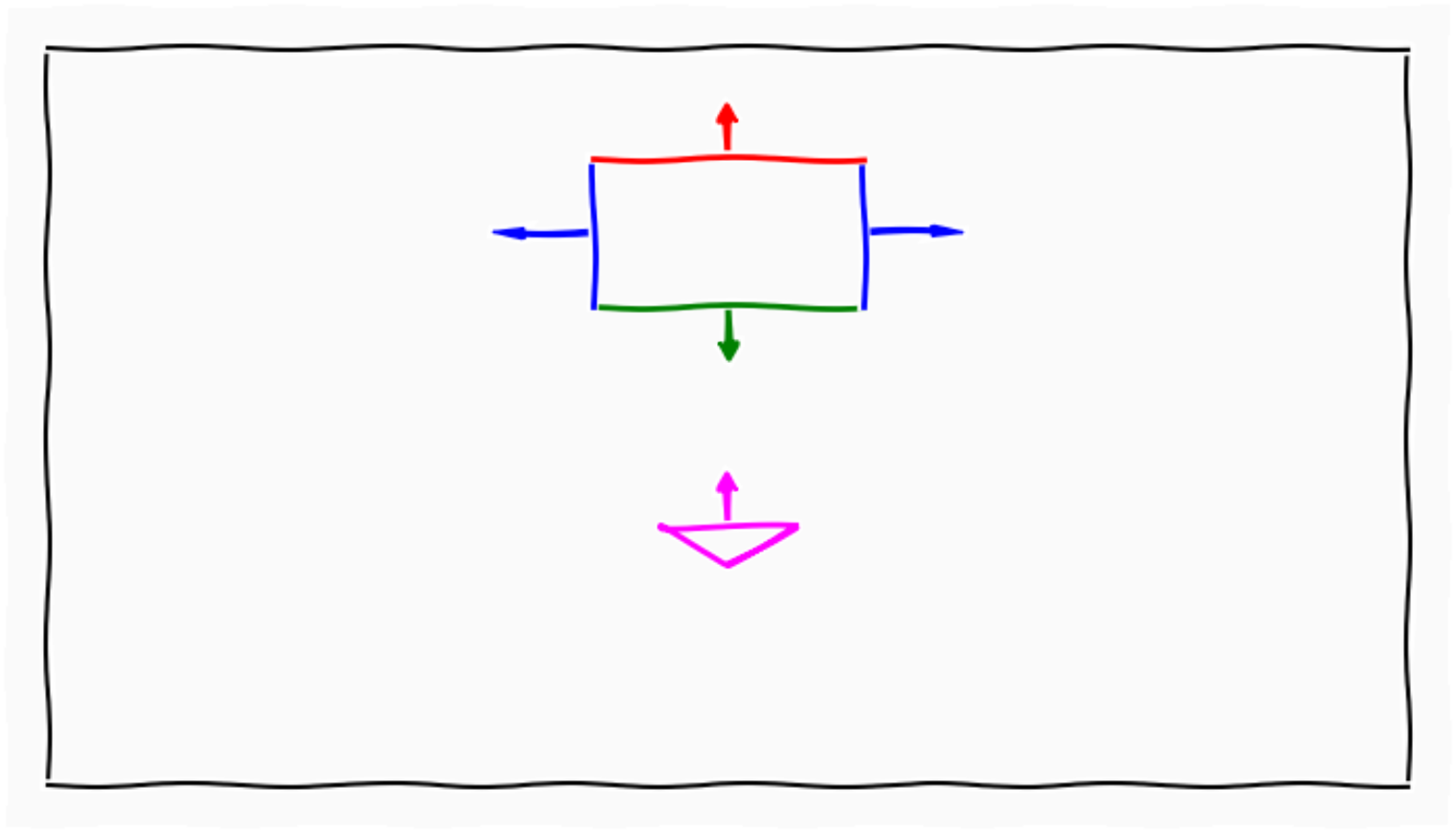
```
for(int i=0; i< NR_TRIANGLES)
  {
    for(int j=0; j<3; j++)
      {
        /* compute vertex normal */
        /* compute shading */
      }
  }
```

Depth Buffer

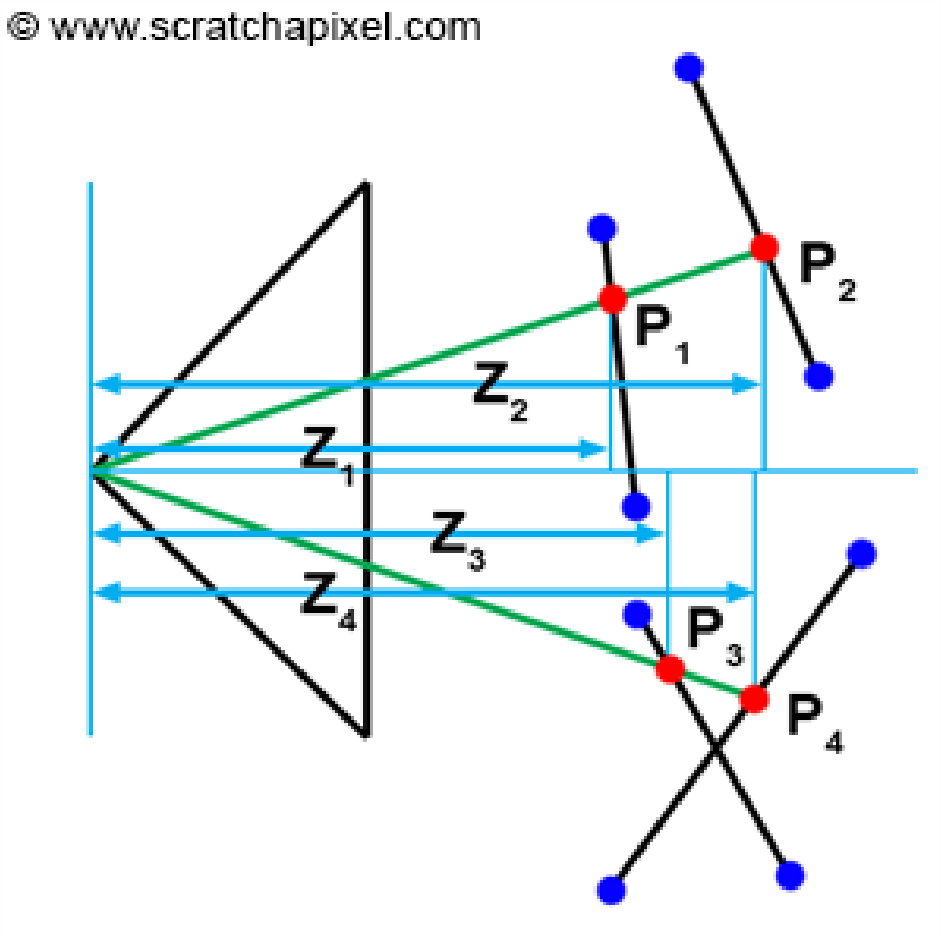
Visibility



Culling



Visibility



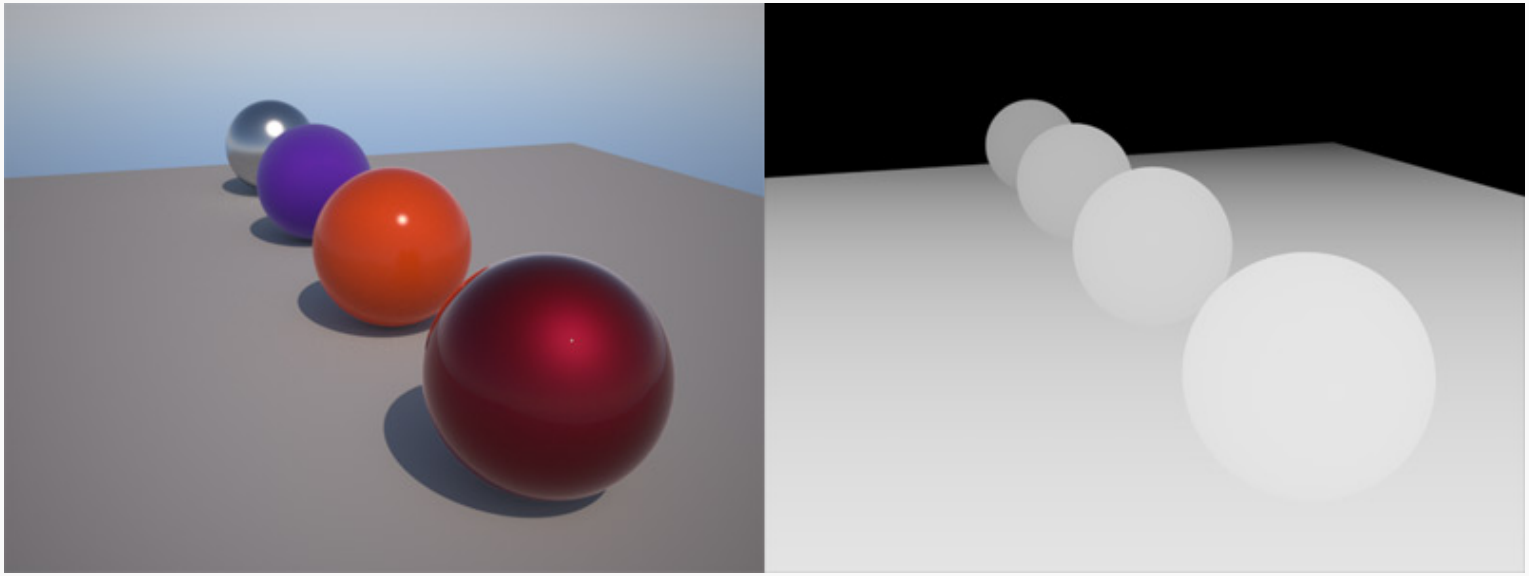
Depth Buffer

1. Create buffer same size as screen
2. Initialise buffer with `max` type value
3. Loop through all primitives
4. If current `z` is smaller than depth value
    - $\Rightarrow$ draw pixel
    - Update depthbuffer with new `z`

Code

```
uint32_t* depthbuffer = (uint32_t)
  malloc(sizeof(unint32_t)*width*height);
memset(depthbuffer,
       std::numerical_limits<uint32_t>::max(),
       sizeof(unint32_t)*width*height);
for(unit32_t i=0;i<nr_triangles;i++)
  {
    if(z<depthbuffer[v*width+u])
      {
        depthbuffer[v*width+u] = z;
        /* draw pixel */
      }
  }
```

Depth Buffer



Fog

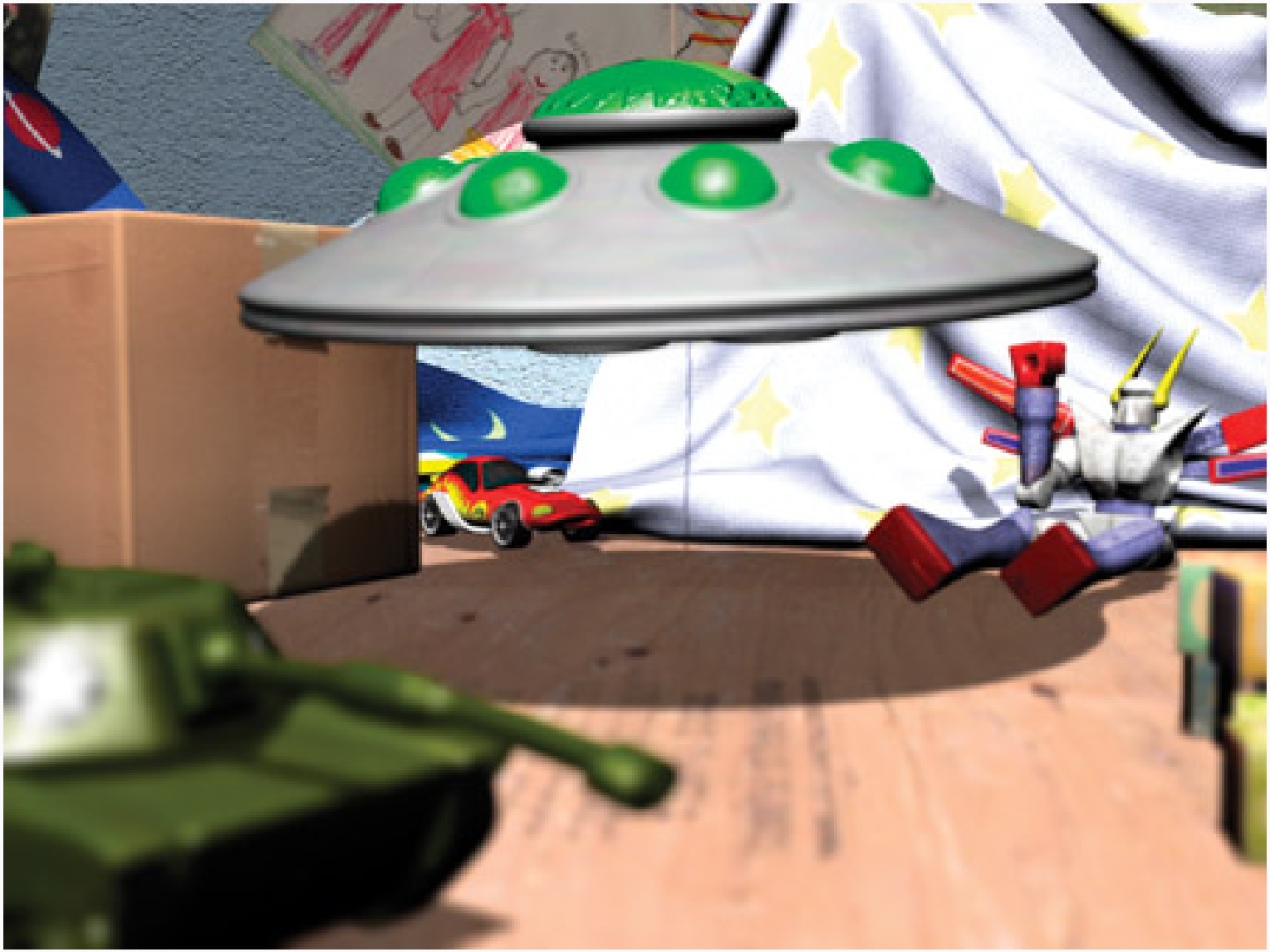
Fog

$$c(u,v) = (1 - \lambda(u,v)) \cdot c^{\text{shading}}(u,v) + \lambda(u,v) c^{\text{fog}}(u,v)$$

$$\lambda_a = \max\left(\frac{r_a - r_0^{\text{fog}}}{r_1^{\text{fog}} - r_0^{\text{fog}}}, 0\right)$$

- We have already generated a depth buffer
- Blend pixels with a "fog" based on depth
- Compute blending at each vertex and interpolate out

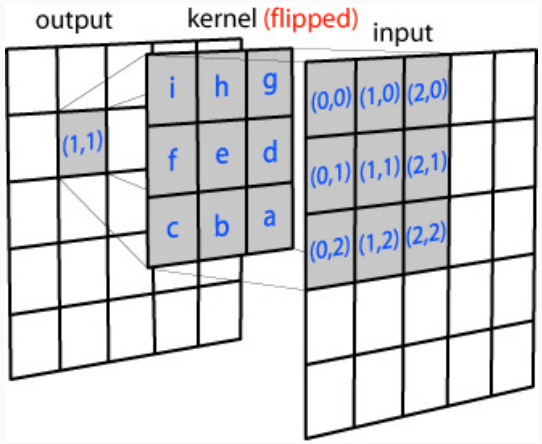
Depth of Field



Depth of Field

$$c(u,v) = \sum_{-1}^{1}\sum_{-1}^{1} k^z(x,y) \cdot c(u+x, v+y)$$

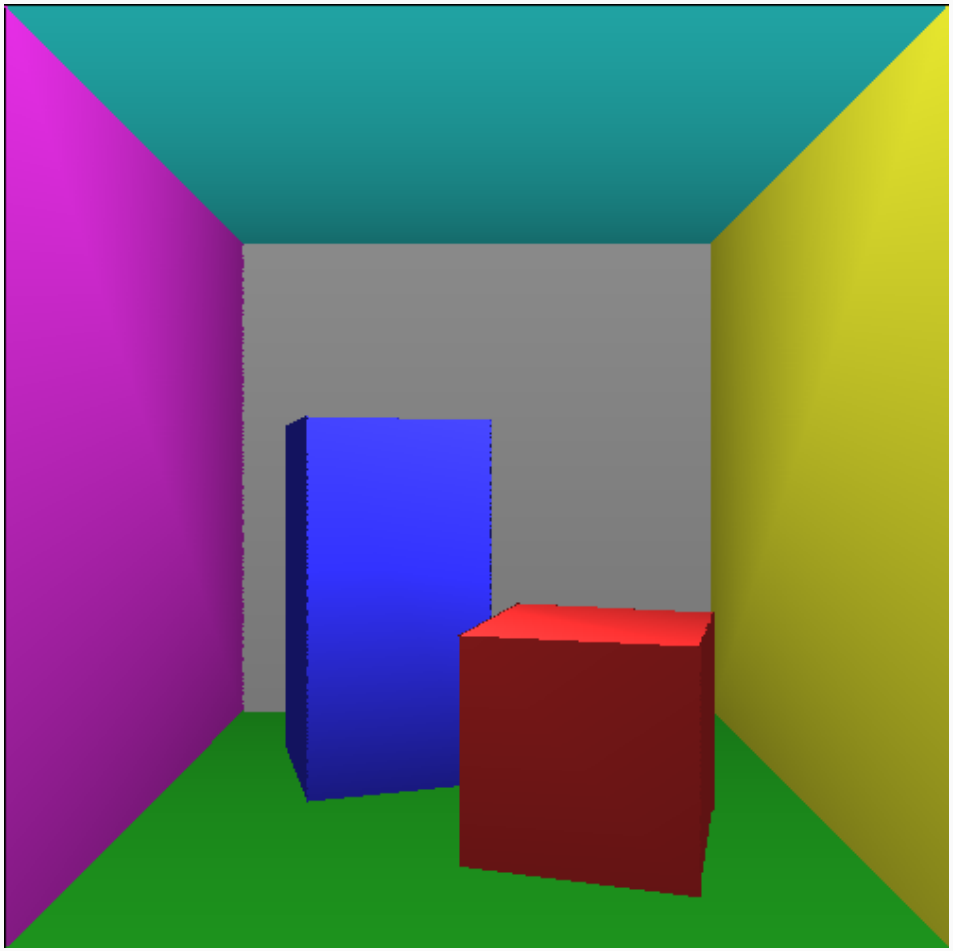
- Blur image based on depth
- Make the kernel dependent on depth



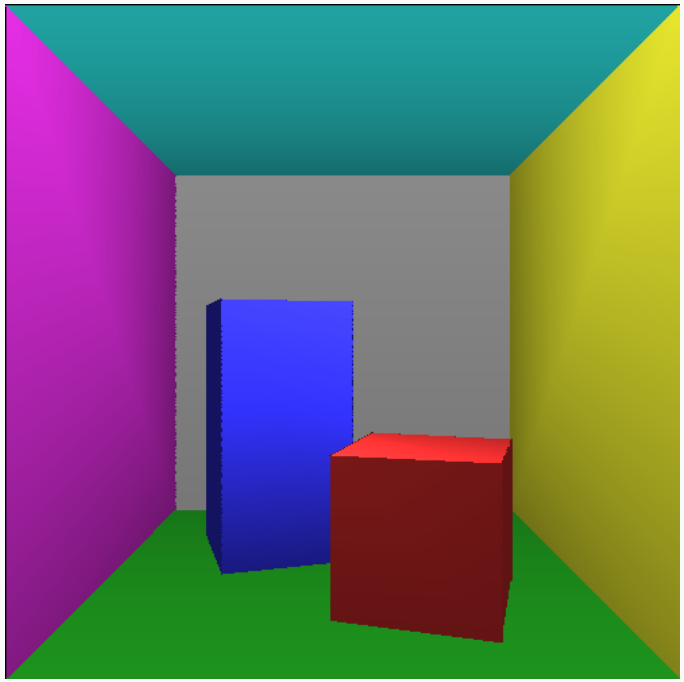
Ideas

- manipulating the normals based on depth?
- manipulating shading based on depth
  - flat shade polygons far away
- special treatment for areas with large depth change
  - finite-difference of z-buffer is an approximation to depth gradient
- make other things a function of depth
  - if it looks good its correct!!

Aliasing



Aliasing



Summary

Summary

- Interpolation is the key to speed-up rendering
- Vertex shading vs. Pixel shading
- We have left the realms of physics, interpolate stuff and fake effects
  - fog, depth-of-field, aa, etc. etc.
  - come up with new things, play around, innovate

Next Time

Lecture textures

- Mapping
  - texture mapping
  - mipp-mapping
  - anti-aliasing
- Buffers
  - Stencil
  - Accumulation
- Shadows

Lab continue with Lab 2

eof