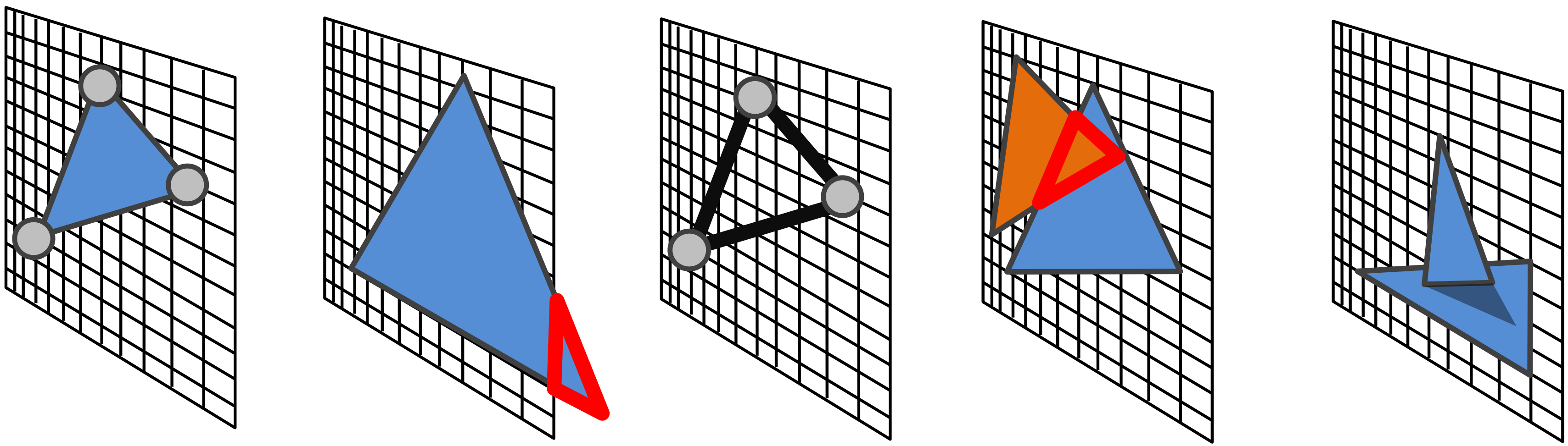
Computer Graphics (COMP0027) 2022/23

Interpolation and z-buffering

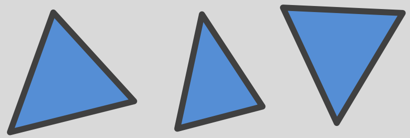
Tobias Ritschel

Challenges



Projection Clipping Rasterization Visibility Shading

Pipeline



Primitives

Projection

Clipping

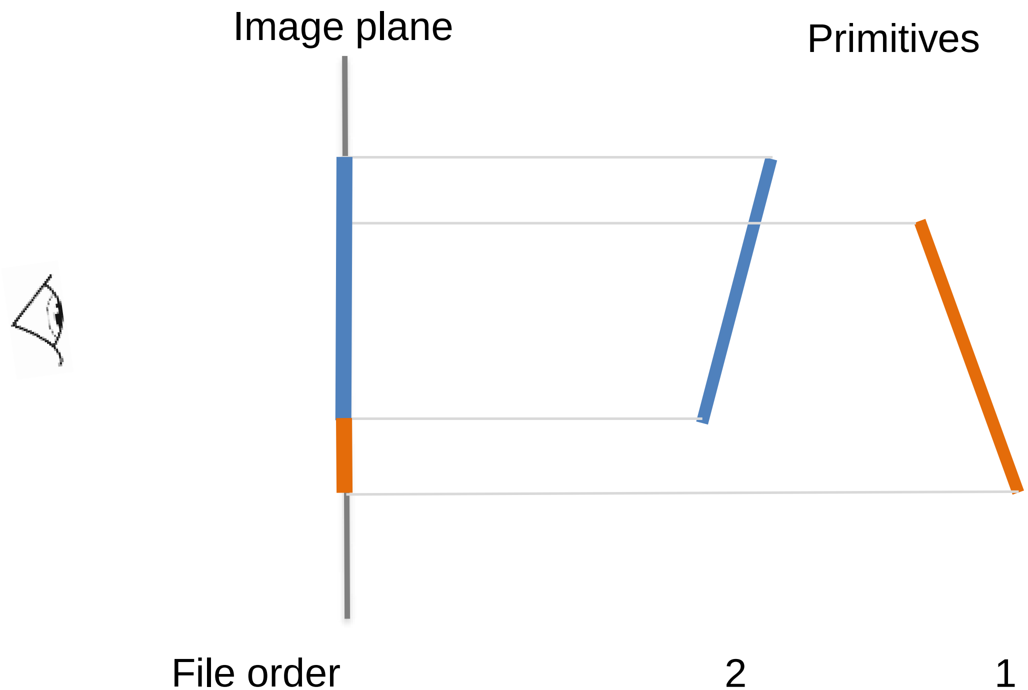
Culling

Rasterisation

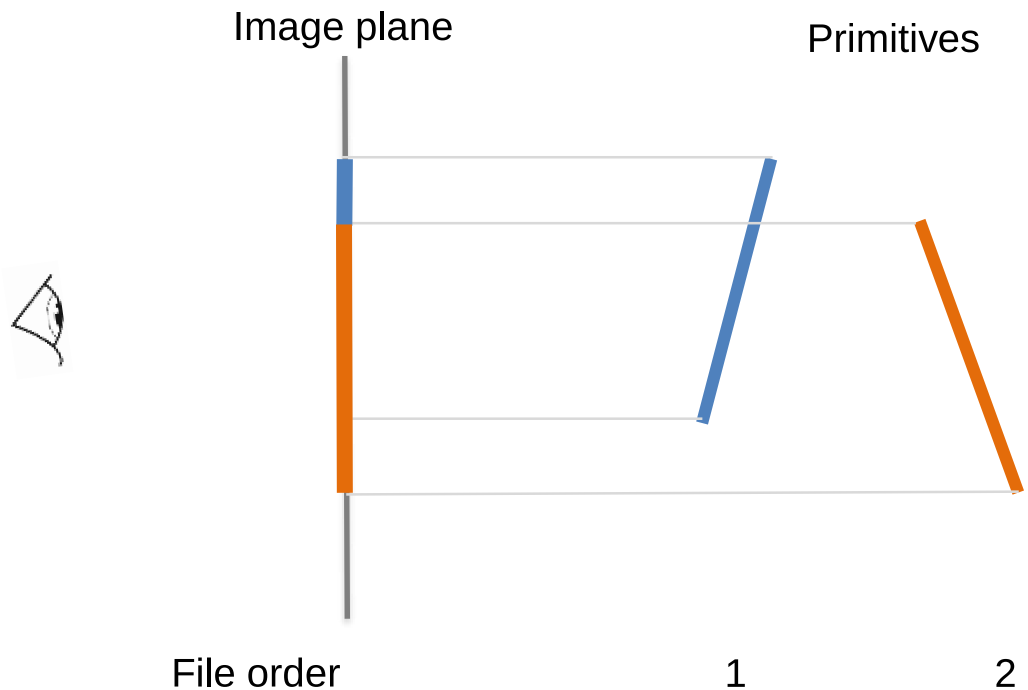
z test

Shading

Doing nothing (with some luck)

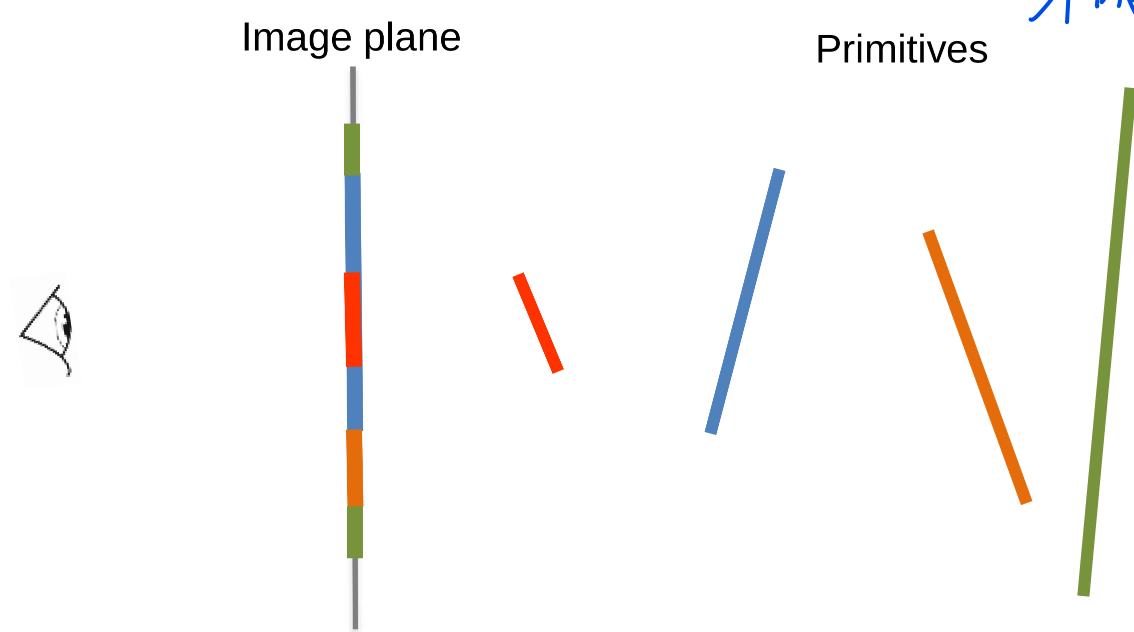


Doing nothing (with no luck)



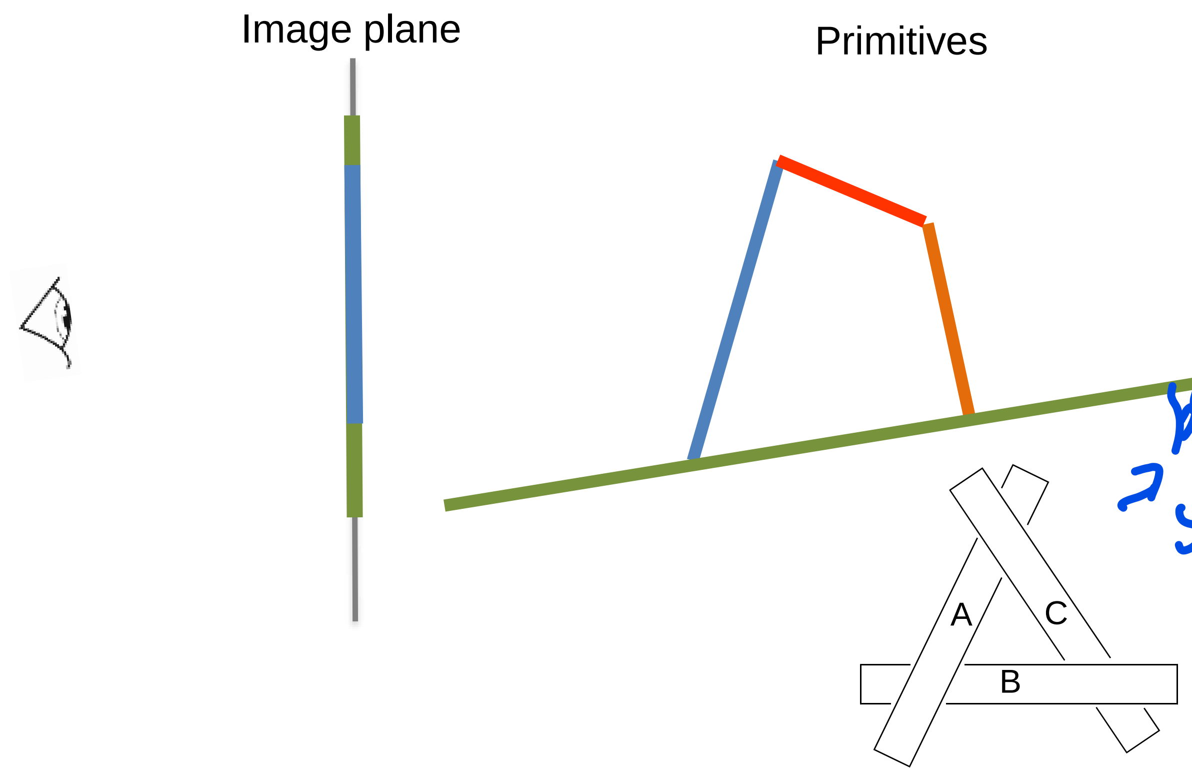
Painters algorithm

从前往后 →
才能防止
overdraw 这



File order	1	4	2	3
Draw order = sort by depth	4	3	2	1

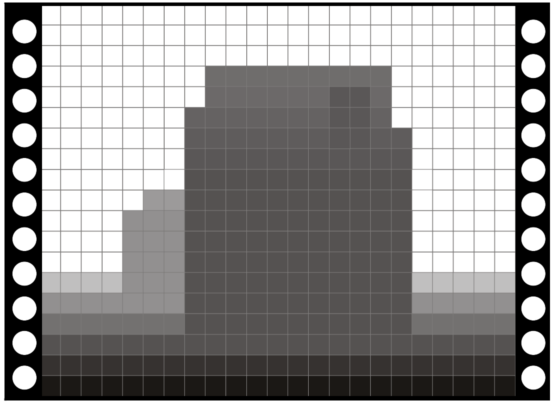
Painters algorithm



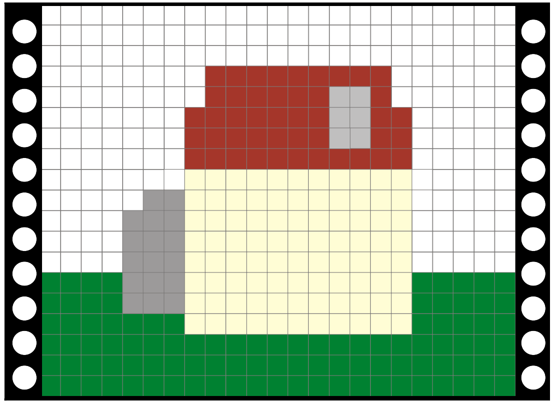
所以没法完全
sort
primitive

z-Buffer

- In addition to color buffer
- Depth buffer (z of view space coordinate)



Depth buffer



Color buffer

Basic Idea

- Initialise the z-Buffer array `zbuffer[width][height]` to z_{max}
- Consider a point at (x, y, z), projected to pixel (x_s, y_s) with colour c
- If $z <$ `zbuf`$[x_s][y_s]$
  – set `colorBufer`$[x_s][y_s] = c$, `zBuffer`$[x_s][y_s] = z$
  – or else, do nothing

Rasterization code with z test

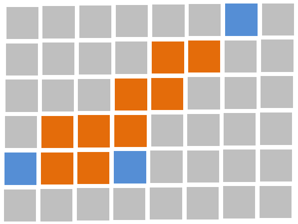
```
For every triangle
 ComputeProjection
 Compute bbox, clip bbox to screen limits
 Compute line equations
 For all pixels in bbox
  If all line equations > 0
   If z < depthBuffer[x, y]{
    framebuffer[x, y] = color;
    depthBuffer[x, y] = depth;
   }
```

Problems

- Aliasing on depth (z-buffer tearing)
- Which z value?
- How to compute z value at each pixel?

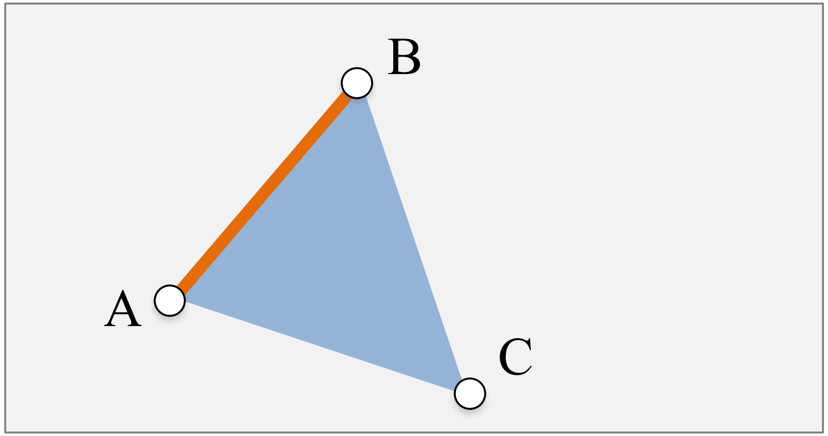
Per-vertex and per-color depth

- Now we have to write a z value for each pixel
- Problem: We only know it at the vertices
- Solution: **Interpolate**
  - We will look at this in more detail!
  - The same interpolation will also give us per-pixel
    - colors,
    - normals,
    - texture coords,
    - etc

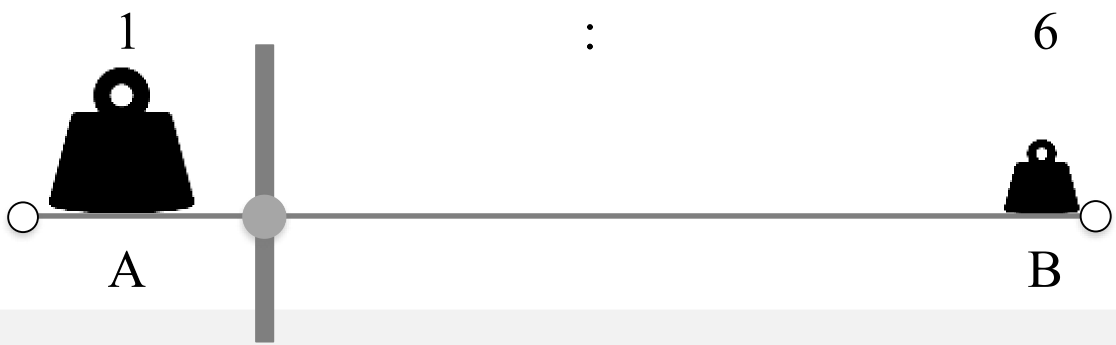


Line vs. area interpolation

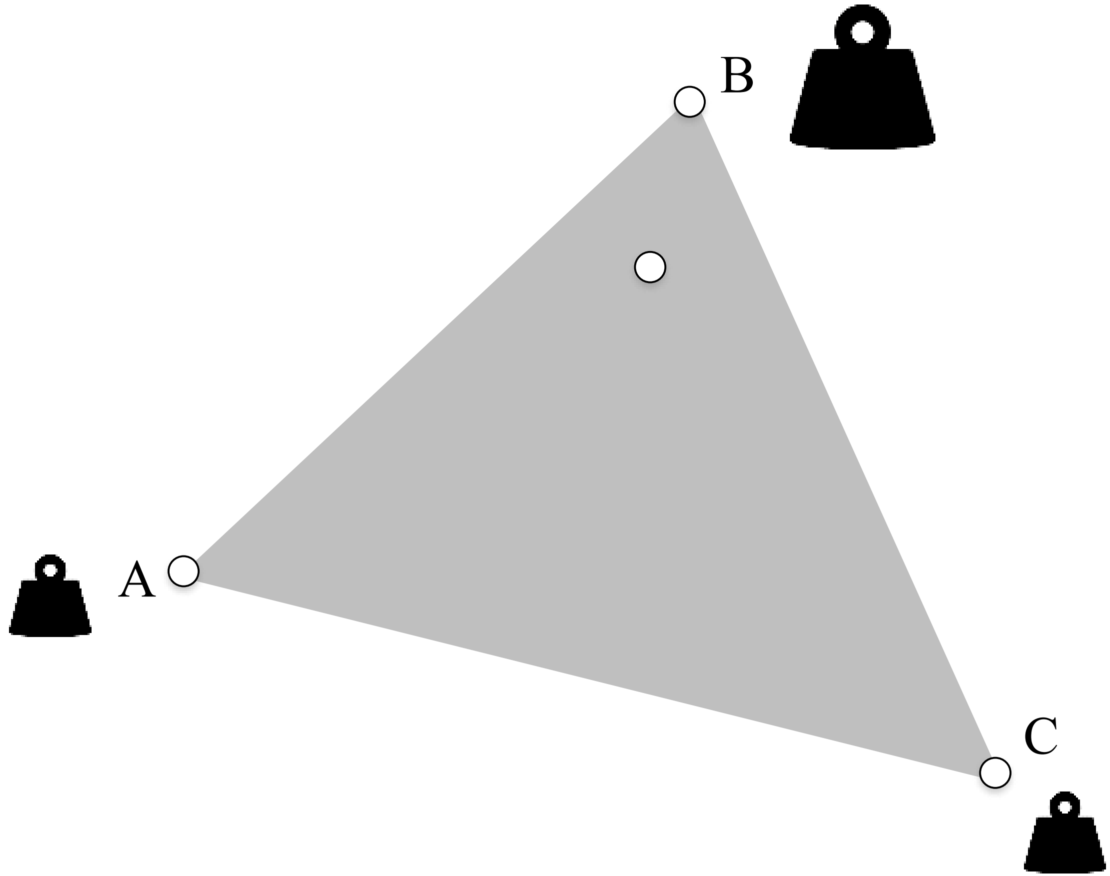
- We now know how to interpolate between **two** points A and B
- But triangle A, B, C has **three** points
- Solution: Barycentric interpolation



Barycentric coordinates



Barycentric coordinates



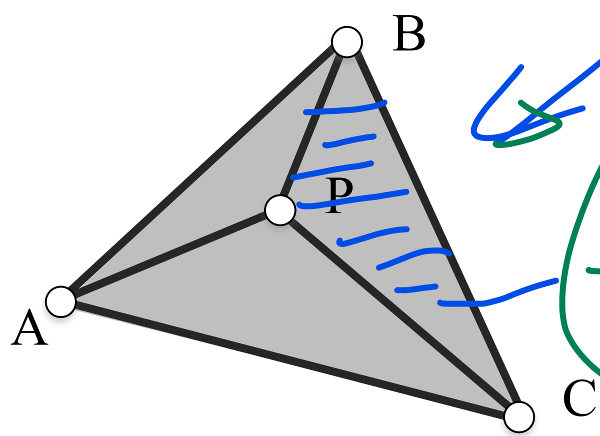
Barycentic coordinates: Definition

因为已经去掉考虑D分配了，所以不需要check正负

$w_A(P) = t(P, B, C) / t(A, B, C)$

$w_B(P) = t(P, C, A) / t(A, B, C)$

$w_C(P) = t(P, A, B) / t(A, B, C)$



$\vec{AP} = x\vec{AB} + (1-x)\vec{AC}$

$\vec{AP} - x\vec{AB} + (x-1)\vec{AC} = 0$

t(A, B, C) is the triangle are

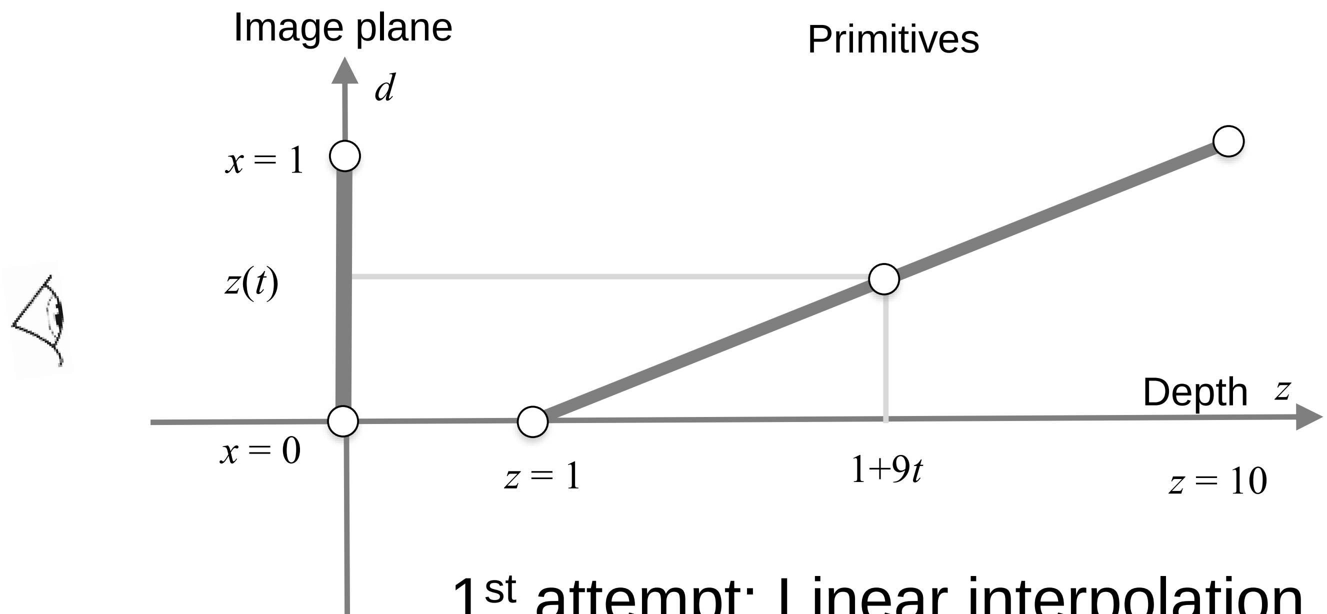
Barycentic coordinates: Definition

$$w_A(P) = t(P, B, C) / t(A, B, C)$$
$$w_B(P) = t(P, C, A) / t(A, B, C)$$
$$w_C(P) = t(P, A, B) / t(A, B, C)$$

Example:

$$color(P) = color_A w_A(P) + color_B w_B(P) + color_C w_C(P)$$

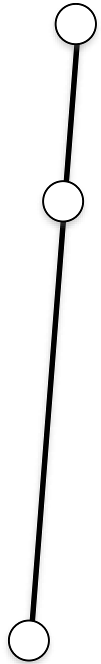
Interpolating z along a line



1st attempt: Linear interpolation

Really ?? (正交 orthogonal)

所以上面的
attemp 不行

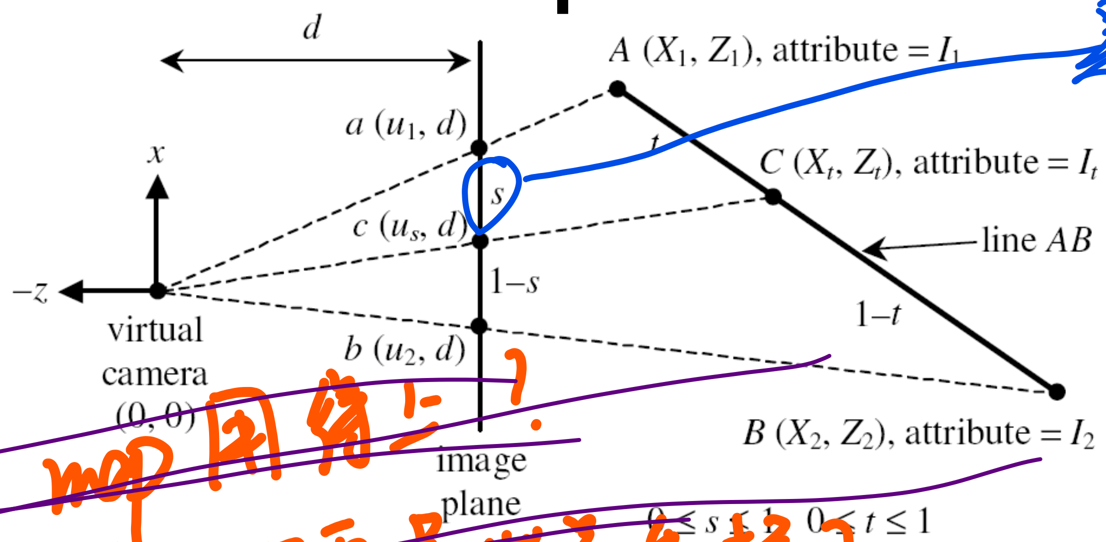


Why is this incorrect?

- Interpolating z linearly in 2d is **incorrect**!
- Why is that?
  - Projection of a point onto screen is done with non-linear projection matrix (Remember: $w = (z + 1)$ factor)
  - **Must** take that into account

Perspective Correct Depth Interpolation

这里是为了透视矫正插值……



是比例
同理会有

~~shadow map 用错了？~~
~~因为z不再是世界坐标了~~

Given this scenario, can write down equations and see that we need to linearly interpolate $1 / z$ (and not z)

这么是个很绕有的问题，project之前加一维保存深度

z ||

这里实求的不是u_s而是它和
u_s求z_t

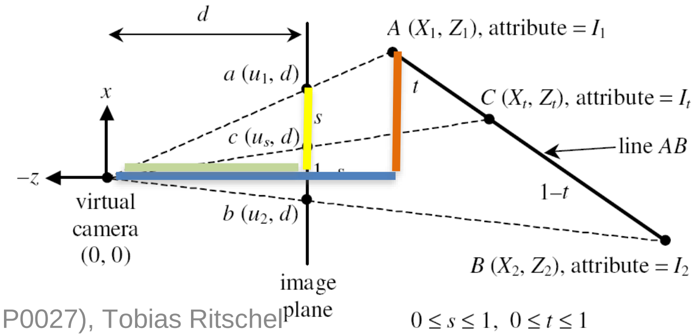
Perspective Correct Depth Interpolation

By similar triangles we have:

$$\frac{X_1}{Z_1} = \frac{u_1}{d} \Rightarrow X_1 = \frac{u_1 Z_1}{d}, \qquad (1)$$

$$\frac{X_2}{Z_2} = \frac{u_2}{d} \Rightarrow X_2 = \frac{u_2 Z_2}{d}, \qquad (2)$$

$$\frac{X_t}{Z_t} = \frac{u_s}{d} \Rightarrow Z_t = \frac{dX_t}{u_s}. \qquad (3)$$



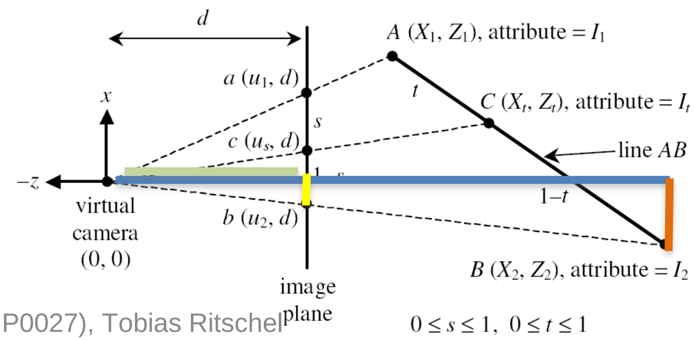
Perspective Correct Depth Interpolation

By similar triangles we have:

$$\frac{X_1}{Z_1} = \frac{u_1}{d} \Rightarrow X_1 = \frac{u_1 Z_1}{d}, \qquad (1)$$

$$\frac{X_2}{Z_2} = \frac{u_2}{d} \Rightarrow X_2 = \frac{u_2 Z_2}{d}, \qquad (2)$$

$$\frac{X_t}{Z_t} = \frac{u_s}{d} \Rightarrow Z_t = \frac{d X_t}{u_s}. \qquad (3)$$



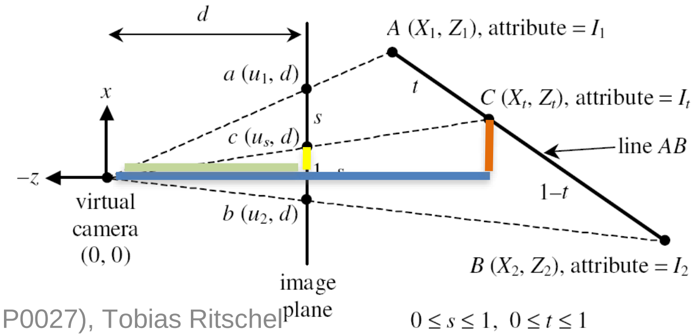
Perspective Correct Depth Interpolation

By similar triangles we have:

$$\frac{X_1}{Z_1} = \frac{u_1}{d} \Rightarrow X_1 = \frac{u_1 Z_1}{d}, \qquad (1)$$

$$\frac{X_2}{Z_2} = \frac{u_2}{d} \Rightarrow X_2 = \frac{u_2 Z_2}{d}, \qquad (2)$$

$$\frac{X_t}{Z_t} = \frac{u_s}{d} \Rightarrow Z_t = \frac{dX_t}{u_s}. \qquad (3)$$

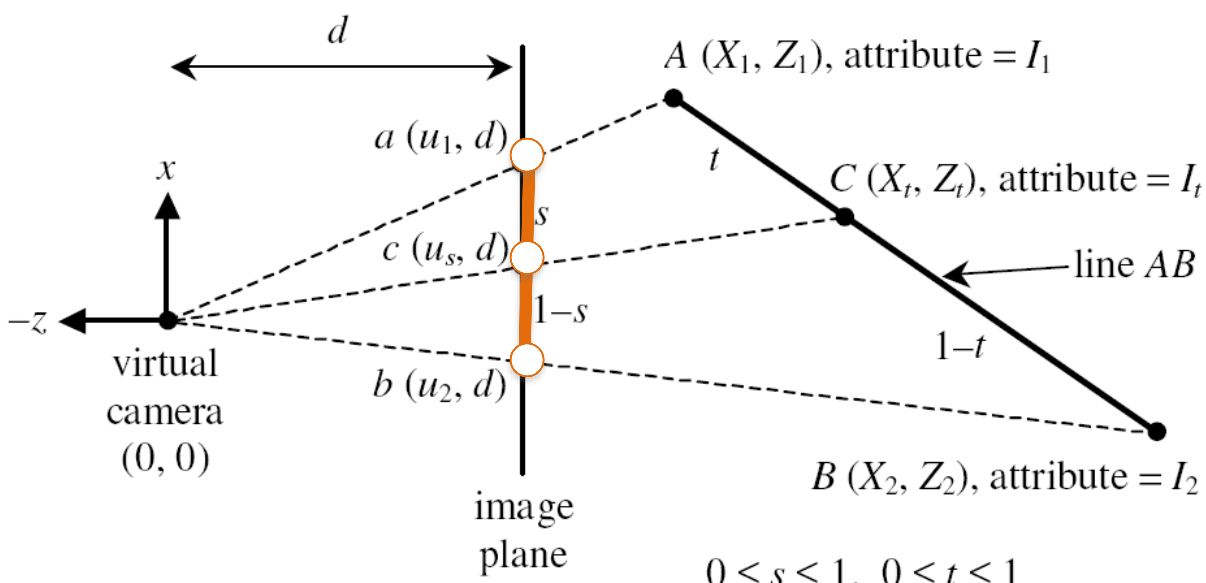


$0 \le s \le 1, \ 0 \le t \le 1$

Perspective Correct Depth Interpolation

By linearly interpolating in the image plane (or screen space), we have

$$u_s = u_1 + s(u_2 - u_1). \tag{4}$$

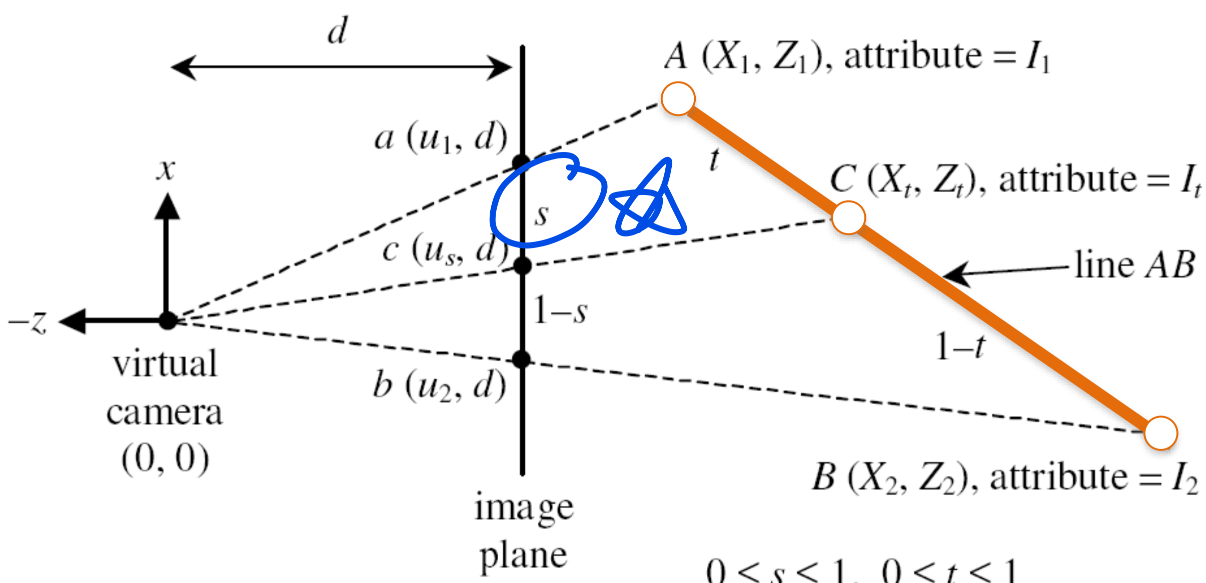


Perspective Correct Depth Interpolation

By linearly interpolating across the primitive in the camera coordinate system, we have

$$X_t = X_1 + t(X_2 - X_1)\,, \tag{5}$$

$$Z_t = Z_1 + t(Z_2 - Z_1)\,, \tag{6}$$



Perspective Correct Depth Interpolation

By linearly interpolating in the image plane (or screen space), we have

$$u_s = u_1 + s(u_2 - u_1)\,. \tag{4}$$

By linearly interpolating across the primitive in the camera coordinate system, we have

$$X_t = X_1 + t(X_2 - X_1)\,, \tag{5}$$

$$Z_t = Z_1 + t(Z_2 - Z_1)\,, \tag{6}$$

Substituting (4) and (5) into (3),

$$Z_t = \frac{d\left(X_1 + t\left(X_2 - X_1\right)\right)}{u_1 + s(u_2 - u_1)}\,. \tag{7}$$

Perspective Correct Depth Interpolation

Substituting (1) and (2) into (7),

$$Z_t = \frac{d\left(\frac{u_1 Z_1}{d} + t\left(\frac{u_2 Z_2}{d} - \frac{u_1 Z_1}{d}\right)\right)}{u_1 + s(u_2 - u_1)}$$

$$= \frac{u_1 Z_1 + t(u_2 Z_2 - u_1 Z_1)}{u_1 + s(u_2 - u_1)}. \qquad (8)$$

Substituting (6) into (8),

$$Z_1 + t(Z_2 - Z_1) = \frac{u_1 Z_1 + t(u_2 Z_2 - u_1 Z_1)}{u_1 + s(u_2 - u_1)}, \qquad (9)$$

Perspective Correct Depth Interpolation

which can be simplified into

$$t = \frac{sZ_1}{sZ_1 + (1-s)Z_2}. \tag{10}$$

Substituting (10) into (6), we have

$$Z_t = Z_1 + \frac{sZ_1}{sZ_1 + (1-s)Z_2}(Z_2 - Z_1), \tag{11}$$

which can be simplified to

$$Z_t = \frac{1}{\frac{1}{Z_1} + s\left(\frac{1}{Z_2} - \frac{1}{Z_1}\right)}. \tag{12}$$

“Can be simplified” in Eq. 10

z1 + t(z2 - z1) = (u1z1 + t(u2z2 - u1z1)) / (u1 + s(u2 - u1))
z1 + tz2 - tz1 = (u1z1 + t(u2z2 - u1z1)) / (u1 + s(u2 - u1))
0 = (u1z1 + t(u2z2 - u1z1)) / (u1 + s(u2 - u1)) - z1 - tz2 + tz1
0 = u1z1 + t(u2z2 - u1z1)) + (- z1 - tz2 + tz1) * (u1 + s(u2 - u1)
0 = u1z1 + t(u2z2 - u1z1) + (- z1(u1 + s(u2 - u1) – tz2(u1 + s(u2 - u1) + tz1(u1 + s(u2 - u1))
0 = u1z1 + tu2z2 - tu1z1 - z1u1 - z1s(u2 - u1) - tz2u1 - tz2s(u2 - u1) + tz1u1 + tz1s(u2 - u1)
0 = tu2z2 - z1s(u2 - u1) - tz2u1 - tz2s(u2 - u1) + tz1s(u2 - u1)
0 = tu2z2 - z1su2 + z1su1 - tz2u1 - tz2su2 + tz2su1 + tz1su2 - tz1su1
0 = - z1su2 + z1su1 + tu2z2 - tz2u1 - tz2su2 + tz2su1 + tz1su2 - tz1su1
0 = - z1su2 + z1su1 + t(u2z2 - z2u1 - z2su2 + z2su1 + z1su2 - z1su1)
t = z1su2 - z1su1 / (u2z2 - z2u1 - z2su2 + z2su1 + z1su2 - z1su1)
t = (z1s(u2 - u1)) / (u2z2 - z2u1 - z2su2 + z2su1 + z1su2 - z1su1)
t = (z1s(u2 - u1)) / ((u2 - u1)(z2 - z2s + z1s))
t = z1s / (z2 - z2s + z1s)
t = z1s / (z2(1 - s) + z1s)
t = sz1 / (z1s + (1 - s)z2)

Perspective Correct Depth Interpolation

Thus we only need to linearly interpolate between $1/z$ values:

$$Z_t = \frac{1}{\frac{1}{Z_1} + s\left(\frac{1}{Z_2} - \frac{1}{Z_1}\right)}. \tag{12}$$

Trade-Offs

- Painter's algorithm:
  - Expensive
- z-Buffer can be inaccurate with few bits
  - Really simple to implement though!
- z-Buffer good for small, sparse polygons
- Why doing both can be good?

Recap: Rasterization

- Simple: Painter’s method
- Used today: z-Buffer
- Need to interpolate z across triangles
- Need to do this perspective-correct