Projections

Brian Curless
CSE 557
Fall 2009

Reading

Required:

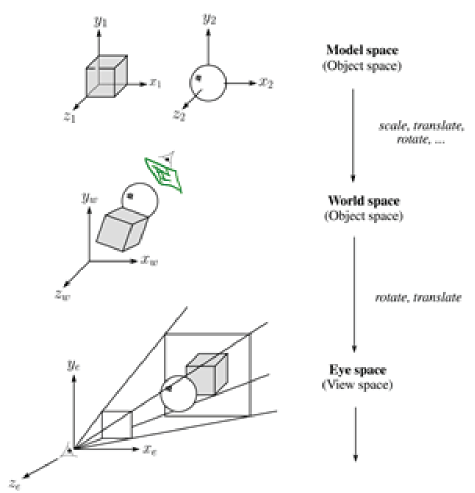
- Shirley, Ch. 7, Sec. 8.2

Further reading:

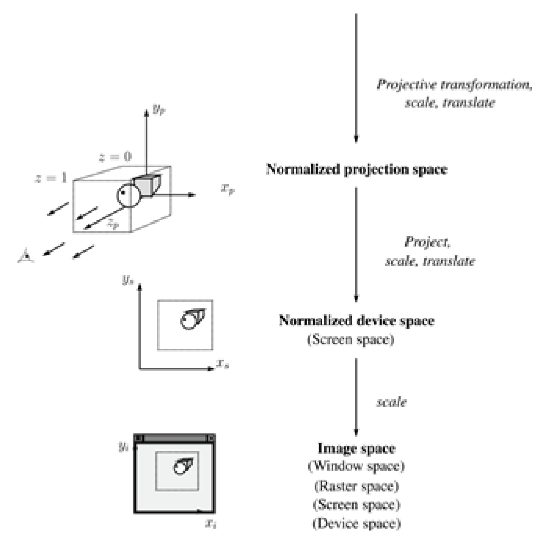
- Foley, et al, Chapter 5.6 and Chapter 6
- David F. Rogers and J. Alan Adams, *Mathematical Elements for Computer Graphics*, 2nd Ed., McGraw-Hill, New York, 1990, Chapter 2.
- I. E. Sutherland, R. F. Sproull, and R. A. Schumacker, A characterization of ten hidden surface algorithms, *ACM Computing Surveys* 6(1): 1-55, March 1974.

3D Geometry Pipeline

Before being turned into pixels by graphics hardware, a piece of geometry goes through a number of transformations...



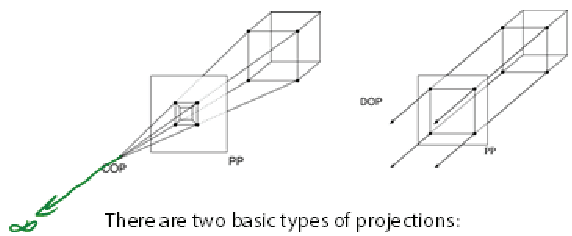
3D Geometry Pipeline (cont'd)



Projections

Projections transform points in n-space to m-space, where $m<n$.

In 3-D, we map points from 3-space to the **projection plane** (PP) (a.k.a., image plane) along **projectors** (a.k.a., viewing rays) emanating from the center of projection (COP):

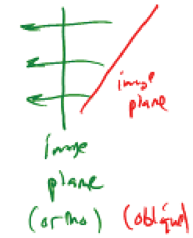


There are two basic types of projections:

- Perspective – distance from COP to PP finite
- Parallel – distance from COP to PP infinite

Parallel projections

For parallel projections, we specify a **direction of projection** (DOP) instead of a COP.



There are two types of parallel projections:

- **Orthographic projection** – DOP perpendicular to PP
- **Oblique projection** – DOP not perpendicular to PP

We can write orthographic projection onto the $z=0$ plane with a simple matrix.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Normally, we do not drop the z value right away. Why not?

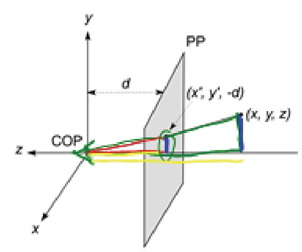
Properties of parallel projection

Properties of parallel projection:

- Not realistic looking
- Good for exact measurements
- Are actually a kind of affine transformation
  - Parallel lines remain parallel
  - Angles not (in general) preserved
- Most often used in CAD, architectural drawings, etc., where taking exact measurement is important

Derivation of perspective projection

Consider the projection of a point onto the projection plane:



By similar triangles, we can compute how much the x and y coordinates are scaled:

$$\frac{y'}{y} = \frac{-d}{z} \qquad y' = \frac{-d}{z} y$$

$$x' = \frac{-d}{z} x$$

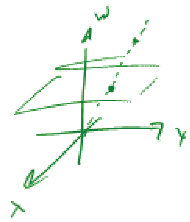
Homogeneous coordinates revisited

Remember how we said that affine transformations work with the last coordinate always set to one.

What happens if the coordinate is not one?

We divide all the coordinates by w:

$$\begin{bmatrix} x \\ y \\ z \\ \underline{w} \end{bmatrix} \rightarrow \begin{bmatrix} x/w \\ y/w \\ z/w \\ 1 \end{bmatrix}$$



If $w = 1$, then nothing changes.

Sometimes we call this division step the "perspective divide."

Homogeneous coordinates and perspective projection

Now we can re-write the perspective projection as a matrix equation:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ -z/d \end{bmatrix}$$

After division by w, we get:

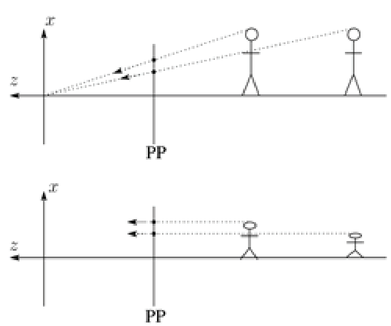
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{x}{z}d \\ -\frac{y}{z}d \\ 1 \end{bmatrix} = \begin{bmatrix} -\frac{d}{z}x \\ -\frac{d}{z}y \\ 1 \end{bmatrix}$$

Again, projection implies dropping the z coordinate to give a 2D image, but we usually keep it around a little while longer.

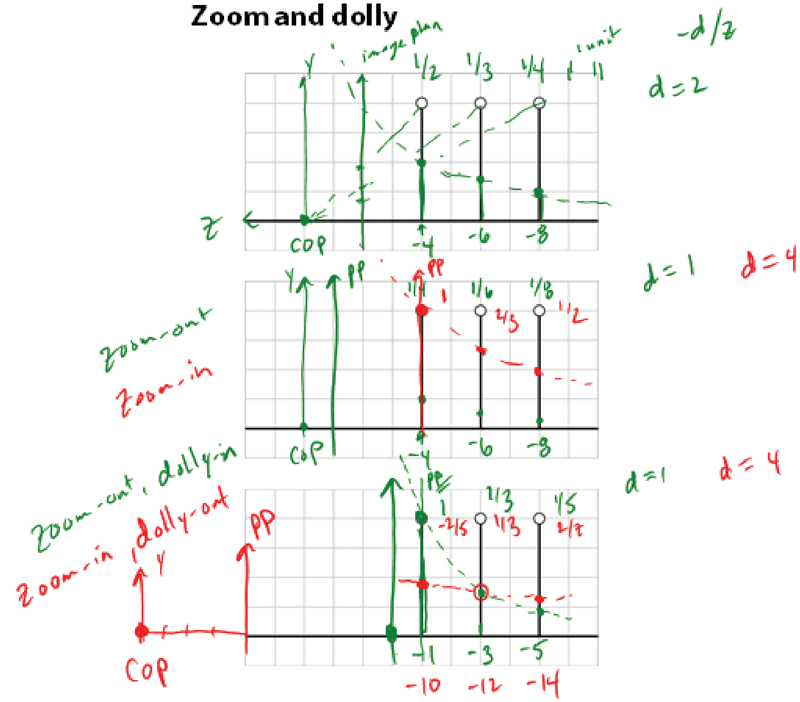
Projective normalization

After applying the perspective transformation and dividing by w, we are free to do a simple parallel projection to get the 2D image.

What does this imply about the shape of things after the perspective transformation + divide?



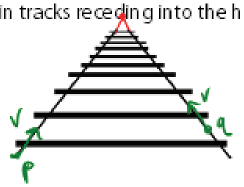
Zoom and dolly



Vanishing points

What happens to two parallel lines that are not parallel to the projection plane?

Think of train tracks receding into the horizon...



The equation for a line is:

$$\mathbf{l} = \mathbf{p} + t\mathbf{v} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} + t \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} = \begin{bmatrix} p_x + tv_x \\ p_y + tv_y \\ p_z + tv_z \\ 1 \end{bmatrix}$$

After perspective transformation we get:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} p_x + tv_x \\ p_y + tv_y \\ -(p_z + tv_z)/d \end{bmatrix}$$

13

Vanishing points (cont'd)

Dividing by w:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} -\dfrac{p_x + tv_x}{p_z + tv_z} d \\ -\dfrac{p_y + tv_y}{p_z + tv_z} d \\ 1 \end{bmatrix}$$

Letting t go to infinity:

$$x'_\infty = \lim_{t \to \infty} -\frac{p_x + tv_x}{p_z + tv_z} d = -\frac{v_x}{v_z} d$$

$$y'_\infty = -\frac{v_y}{v_z} d$$

We get a point!

What happens to the line $\mathbf{l} = \mathbf{q} + t\mathbf{v}$?

Each set of parallel lines intersect at a **vanishing point** on the PP.

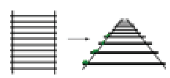
Q: How many vanishing points are there? ∞

14

Properties of perspective projections

The perspective projection is an example of a **projective transformation**.

Here are some properties of projective transformations:



- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Ratios are <u>not</u> preserved

One of the advantages of perspective projection is that size varies inversely with distance – looks realistic.

A disadvantage is that we can't judge distances as exactly as we can with parallel projections.

15

Human vision and perspective

$k_d \hat{N} \cdot L$

$L^T \underbrace{k_d \hat{N}}_{N}$

$L_1^T N = I_1$

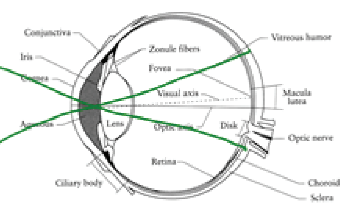
$L_2^T N = I_2$

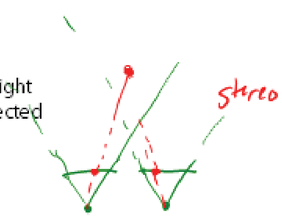
$L_3^T N = I_3$

$\begin{bmatrix} L \end{bmatrix} N = \begin{bmatrix} I \end{bmatrix}$

photometric stereo

The human visual system uses a lens to collect light more efficiently, but records perspectively projected images much like a pinhole camera.





Q: Why did nature give us eyes that perform perspective projections?

size cue, field of view

Q: Do our eyes "see in 3D"?

size cues (priors)

focus

motion parallax

atmosphere effects / fog

shadows

shading

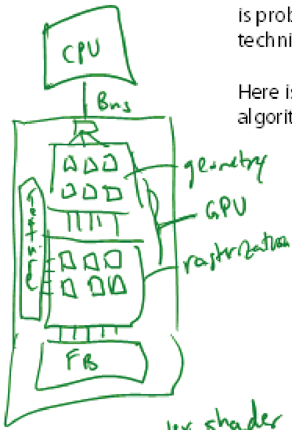
shape from texture

16

Z-buffer

We can use projections for **hidden surface elimination**.

The **Z-buffer'** or **depth buffer** algorithm [Catmull, 1974] is probably the simplest and most widely used of these techniques.



Here is pseudocode for the Z-buffer hidden surface algorithm:

```
for each pixel (i,j) do
    Z-buffer [i,j] ← FAR
    Framebuffer[i,j] ← <background color>
end for
for each polygon A do      ← For each vertex of A, shader xform vertices → A'
    for each pixel in A do
        Compute depth z and shade s of A' at (i,j)
        if z > Z-buffer [i,j] then
            Z-buffer [i,j] ← z
            Framebuffer[i,j] ← s
        end if
    end for
end for
```

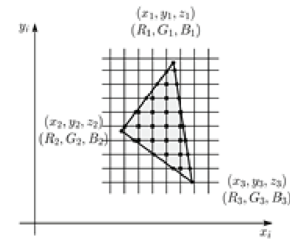
vertex shader
geometry shaders
pixel/fragment shaders

GPGPU

Z-buffer, cont'd

The process of filling in the pixels inside of a polygon is called **rasterization**.

During rasterization, the *z* value and shade *s* can be computed incrementally (fast!).



<u>Curious fact:</u>

- Described as the "brute-force image space algorithm" by [SSS]
- Mentioned only in Appendix B of [SSS] as a point of comparison for <u>huge</u> memories, but written off as totally impractical.

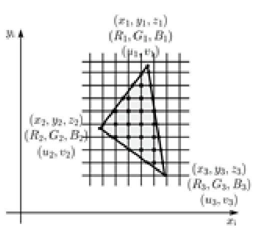
Today, Z-buffers are commonly implemented in hardware.

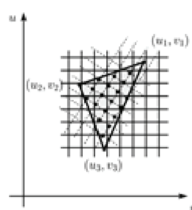
Texture mapping and the z-buffer

Texture-mapping can also be handled in z-buffer algorithms.

Method:

- Scan conversion is done in screen space, as usual
- Each pixel is colored according to the texture
- Texture coordinates are found by Gouraud-style interpolation

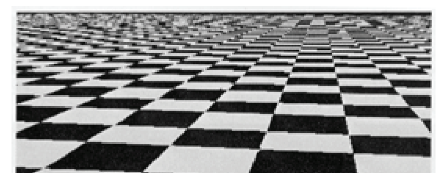




Note: Mapping is more complicated if you want to do perspective right!

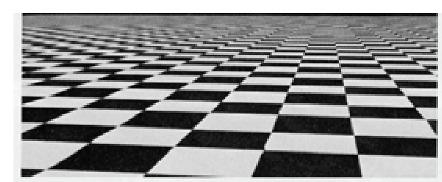
Antialiasing textures

If you render an object with a texture map using point-sampling, you can get aliasing:



From Crow, SIGGRAPH '84

Proper antialiasing requires area averaging over pixels:



From Crow, SIGGRAPH '84

In some cases, you can average directly over the texture pixels to do the anti-aliasing.

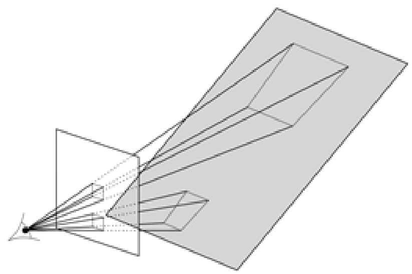
Computing the average color

The computationally difficult part is summing over the covered pixels.

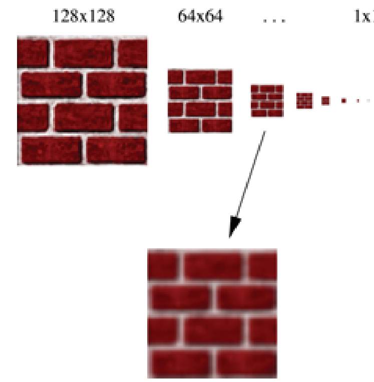
Several methods have been used.

The simplest is **brute force**:

- Figure out which texels are covered and add up their colors to compute the average.



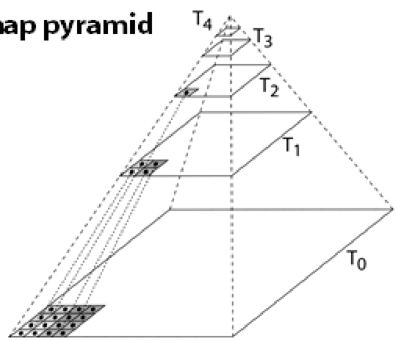
Mip maps



A faster method is **mip maps** developed by Lance Williams in 1983:

- Stands for "multum in parvo" – many things in a small place
- Keep textures prefiltered at multiple resolutions
- Has become the graphics hardware standard

Mip map pyramid



The mip map hierarchy can be thought of as an image pyramid:

- Level 0 ($T_0[i,j]$) is the original image.
- Level 1 ($T_1[i,j]$) averages over 2x2 neighborhoods of original.
- Level 2 ($T_2[i,j]$) averages over 4x4 neighborhoods of original
- Level 3 ($T_3[i,j]$) averages over 8x8 neighborhoods of original

During rendering, a pixel location and its approximate area are used to interpolate among "appropriate" samples in the pyramid.