COMS 30115

Clipping and Culling

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

March 18th, 2019

http://carlhenrik.com

Introduction

Last time

- Mappings
  - Not just textures

Today

- Clipping and Culling

The Book

- Scratchapixel URL
- Blinn and Newell URL
- Fabien Sanglard Webpage URL
- Keneth Joy notes on Clipping URL

Clipping

Our Rasterisation Engine

- We know how to transform geometry
- We know how to project things from 3D space to screen space
- We know how to draw 3D data by interpolation in screen space
- Now we need to figure out what to draw

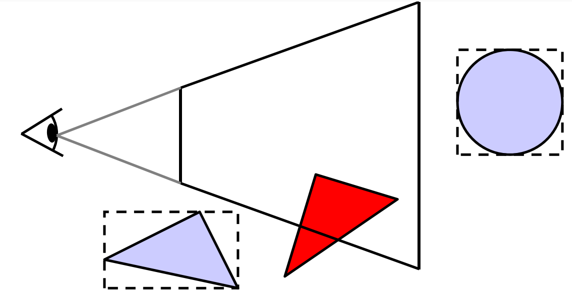
Pipeline

- ☒ Transform world
- ☐ Clip geometry to view Frustrum
- ☒ Project vertices
- ☒ Interpolate depth across polygons
- ☒ Perform depth culling
- ☒ Interpolate shading/textures etc.
- ☒ Perform pixelshading
- ☒ Double buffer
- repeat

Clipping & Culling

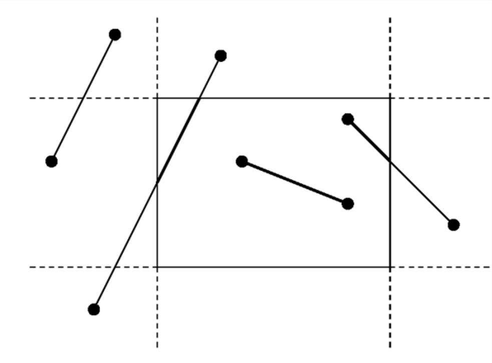
- Drawing each polygon expensive
- Remove elements that we do not need to draw early in the pipeline to save computations
- Culling: remove whole primitive
  - back-face culling, occlusion culling, etc.
- Clipping: remove part of primitive

Clipping & Culling

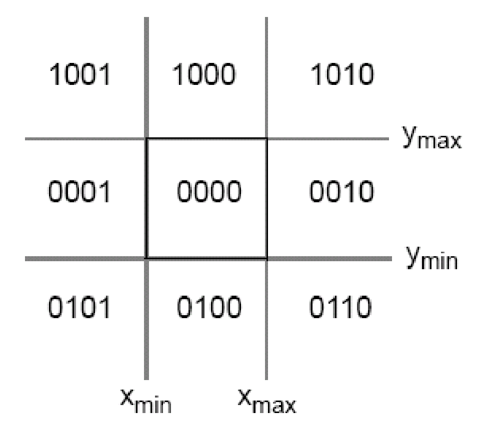


Screen space clipping

Line Clipping



Line Clipping[1]

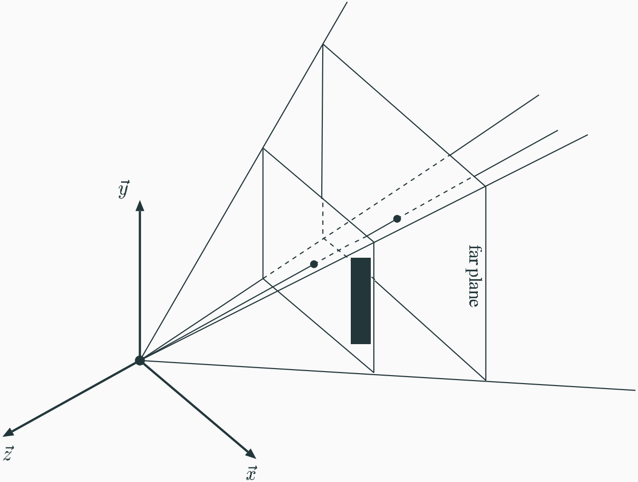


[1] https://en.wikipedia.org/wiki/Cohen%E2%80%93Sutherland_algorithm

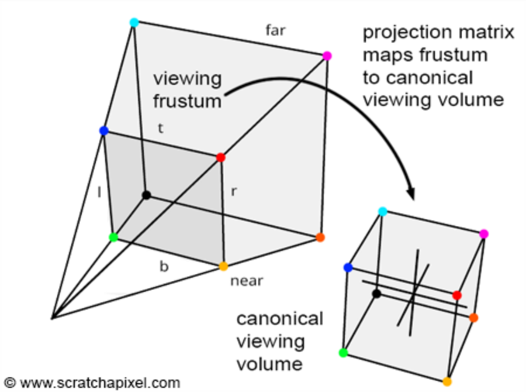
Cohen-Sutherland

1. Compute Outcodes
2. `OR` end-points `BREAK` if `0`
3. `AND` end-points `BREAK` if $\not 0$
4. `AND` end-point with clip-plane
    - `CLIP` if $\not 0$
    - `XOR` end-point with plane

3D Clipping



4D Clipping



far
projection matrix
maps frustum
to canonical
viewing volume
viewing
frustum
t
l
r
b
near
canonical
viewing
volume
© www.scratchapixel.com

Homogenous Coordinates

- We can add a single coordinate w to each point

$$[x, y, z, w]^{\mathrm{T}}$$

- the process of homogenisation is to make $w = 1$ which corresponds to projecting $[x, y, z, w]^{\mathrm{T}}$ to its corresponding point $[\frac{1}{w}x, \frac{1}{w}y, \frac{1}{w}z, 1]^{\mathrm{T}}$ which is a point in 3D space
- this means $[x, y, z, 1]^{\mathrm{T}}$ and $[3x, 3y, 3z, 3]^{\mathrm{T}}$ corresponds to the same point in 3D space

Homogenous Coordinates

Screen space

$$u = \frac{f}{z} \cdot x \quad v = \frac{f}{z} \cdot y$$

Homogenisation

$$\begin{bmatrix} u \\ v \\ f \\ 1 \end{bmatrix} = \frac{f}{z} \begin{bmatrix} x \\ y \\ z \\ \frac{z}{f} \end{bmatrix}$$

Projection Matrix

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

Homogenous Clipping

- We can write the set of all coordinates that coresponds to a screen coordinate with a single homogenous coordinate

$$[x, y, z, \frac{z}{f}]^{\mathrm{T}}$$

- In this space clipping is easy, $x > |w \cdot x_{\mathrm{max}}|$ are all points that should be clipped in x-plane

$$u_{max} = x_{max} \cdot \frac{z}{f}$$

Homogenous Clipping

- Map from world space to clip space

$$[x, y, z, 1]^{\mathrm{T}} \rightarrow [x, y, z, \frac{z}{f}]^{\mathrm{T}}$$

Homogenous Clipping

- Map from world space to clip space

$$[x, y, z, 1]^{\mathrm{T}} \rightarrow [x, y, z, \frac{z}{f}]^{\mathrm{T}}$$

- Clip x and y plane of view frustrum

$$-w \cdot x_{\text{max}} \leq x \leq w \cdot x_{\text{max}}$$
$$-w \cdot y_{\text{max}} \leq y \leq w \cdot y_{\text{max}}$$

Homogenous Clipping

- Map from world space to clip space

$$[x, y, z, 1]^{\mathrm{T}} \rightarrow [x, y, z, \frac{z}{f}]^{\mathrm{T}}$$

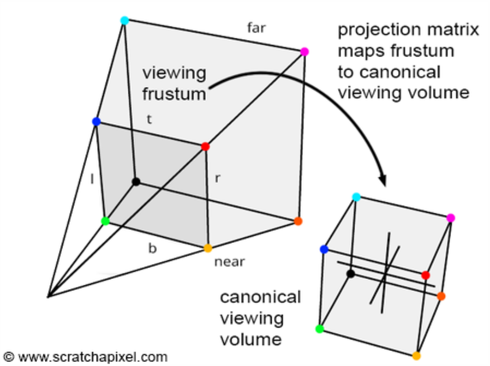
- Clip x and y plane of view frustrum

$$-w \cdot x_{\max} \leq x \leq w \cdot x_{\max}$$
$$-w \cdot y_{\max} \leq y \leq w \cdot y_{\max}$$

- Map homogenous coordinate to screen space by homgenising coordinate

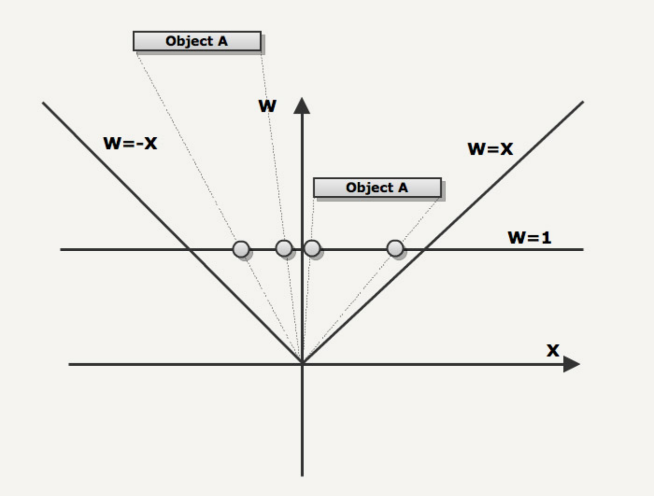
$$[x_{clipped}, y_{clipped}, z, \frac{z}{f}]^{\mathrm{T}} \rightarrow [x_{clipped}\frac{f}{z}, y_{clipped}\frac{f}{z}, z\frac{f}{z}, \frac{z}{f}\frac{f}{z}]^{\mathrm{T}} = [u_{clipped}, v_{clipped}, f, 1]^{\mathrm{T}}$$

3D Clipping[2]



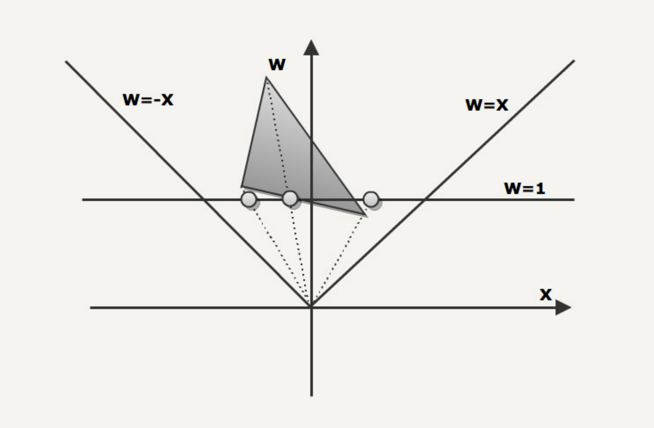
[2] http://fabiensanglard.net/polygon_codec/

3D Clipping[2]



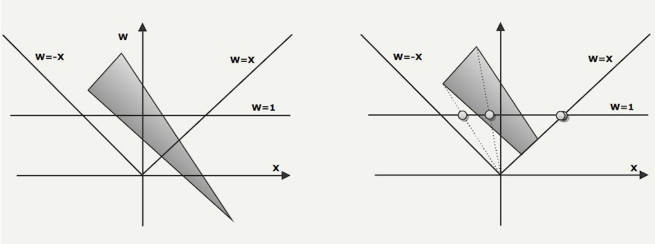
[2] http://fabiensanglard.net/polygon_codec/

3D Clipping[2]



[2] http://fabiensanglard.net/polygon_codec/

3D Clipping[2]

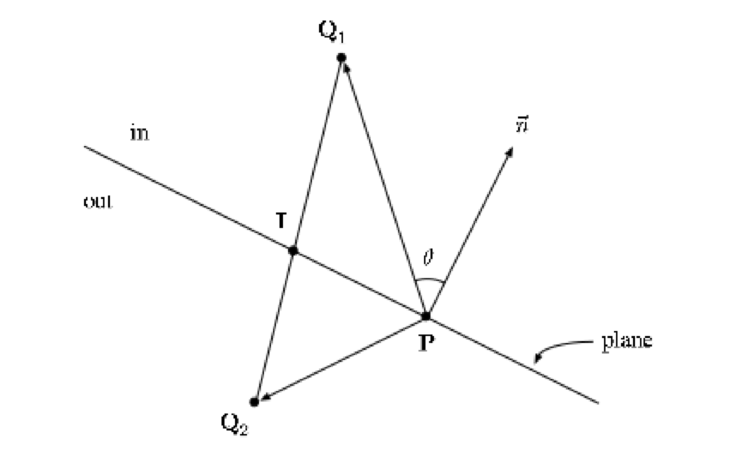


[2] http://fabiensanglard.net/polygon_codec/

Its more than this

- A clipped triangle does not need to be a triangle
  - remap to new triangles
- You need to compute new vertex attributes
  - remember that everything is flat
- Triangles through infinity $\texttt{w} = 0$

Interpolation of Attributes[3]



[3] http://fabiensanglard.net/polygon_codec/clippingdocument/Clipping.pdf

Interpolation of Attributes

$$d_1 = (Q_1 - P) \cdot n$$
$$d_2 = (Q_2 - P) \cdot n$$

Four Cases

- $d_1 \geq 0$ and $d_2 > 0$ or $d_2 \geq 0$ and $d_1 > 0$ Line inside
- $d_1 \leq 0$ and $d_2 < 0$ or $d_2 \leq 0$ and $d_1 < 0$ Line outside
- $d_1 > 0$ and $d_2 < 0$ Q_1 inside and Q_2 outside
- $d_1 < 0$ and $d_2 > 0$ Q_2 inside and Q_1 outside

Interpolation of Attributes

- Intersection point

$$I = Q_1 + t(Q_2 - Q_1)$$

Interpolation of Attributes

- Intersection point

$$I = Q_1 + t(Q_2 - Q_1)$$

- Re-write in terms of P

$$(I - P) = (Q_1 - P) + t\,((Q_2 - P) - (Q_1 - P))$$

Interpolation of Attributes

- Intersection point

$$I = Q_1 + t(Q_2 - Q_1)$$

- Re-write in terms of P

$$(I - P) = (Q_1 - P) + t\left((Q_2 - P) - (Q_1 - P)\right)$$

- Multiply by normal

$$\underbrace{(I - P) \cdot n}_{0} = \underbrace{(Q_1 - P) \cdot n}_{d1} + t \left(\underbrace{(Q_2 - P) \cdot n}_{d2} - \underbrace{(Q_1 - P) \cdot n}_{d1} \right)$$

Interpolation of Attributes

- Intersection point

$$I = Q_1 + t(Q_2 - Q_1)$$

- Re-write in terms of P

$$(I - P) = (Q_1 - P) + t\left((Q_2 - P) - (Q_1 - P)\right)$$

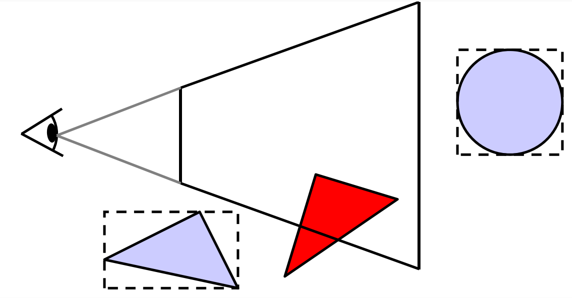
- Multiply by normal

$$\underbrace{(I - P) \cdot n}_{0} = \underbrace{(Q_1 - P) \cdot n}_{d1} + t\left(\underbrace{(Q_2 - P) \cdot n}_{d2} - \underbrace{(Q_1 - P) \cdot n}_{d1}\right)$$

- Solve for t

$$t = \frac{d_1}{d_1 - d_2}$$

Clipping and Culling



Culling

Back-face

- simple, dot product of face normal and view direction positive means not visible
- can we speed up things by clustering normals to remove several directly?

Depth

- z-buffer

Frustrum

- construct bounding boxes and compare
- axis aligned or not?

Extensions

Rasterisation

Visualisations

1. Depth of field
2. Approximate Anti-Aliasing
3. Screen space ambient occlusion
4. Meta-balls and implicit surfaces
5. Shadow maps
6. Stencil Shadows

Techniques

1. Barycentric coordinates
2. Cell shading
3. Normal mapping
4. Texture mapping
5. Mip-mapping
6. Bump mapping
7. Novel lighting

Rasterisation

Clipping

1. Back-face culling
2. Frustrum Culling
3. Frustrum Clipping
4. Screen space clipping

Optimisation

1. SSE & AVX extensions
2. OpenCL
3. OpenMP
4. Framebuffer with memory-aligned PutPixel

Rasterisation

Misc

1. Object Loading
2. Material library
3. Dynamics
    - collision detection
    - "exploding" objects

What I haven't seen

- Fire
- Smoke
- Transparency
- Mirrors
- *Pick any visual phenomenon and think of how to render it*
- Voxels
- Procedural geometry
- etc. etc.

Imagination is the only limitation

Summary

End of Part II

- *Computer* Graphics
    - generate graphics in a manner suitable for commputers
- Sparse (per vertex) computations of light
- Interpolate vertex attributes across pixels (fragments)

End of Part II

- *Computer* Graphics
    - generate graphics in a manner suitable for commputers
- Sparse (per vertex) computations of light
- Interpolate vertex attributes across pixels (fragments)
- We have seen at least one example of each part of the pipeline but there are many many more versions. Choose algorithm based on the hardware that you program.

Rasterisation

- This is how the internals of game engines work
- This is what your GPU does for you (a lot of it at least)
- Now you know how this works
    - my hope have been that this should allow you to make more efficient use of modern APIs
    - understand how and what you can tweak
    - understand how you can exploit things to your benefit
- Fixed Rendering Pipeline is no more

Course so far

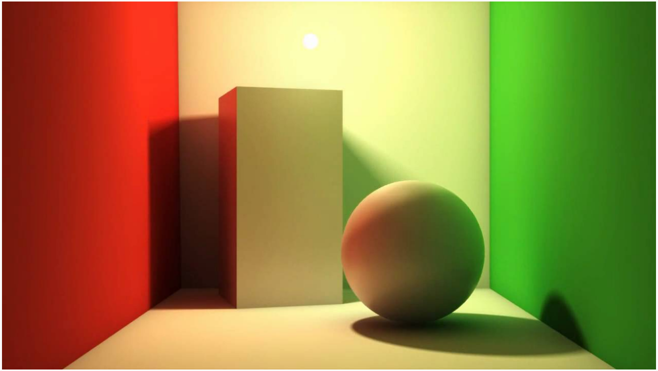
Raytracer how is an image generated

- images are actually quite simple
- only thing holding back realism is computation time

Raster how does a computer display an image

- in lab you have seen what the basics are
- lectures have gone a few steps further

Part II



Part II

$$L(x \to \Theta) = L_e(x \to \Theta) + \int_{\Omega_x} f_r(x, \Psi \to \Theta) L(x \leftarrow \Psi) \cos(\mathbf{n}_x, \Psi) d\omega_\Psi$$

Global Illumination

- What happens when light hits a surface
  - Light comes in reflects and refracts
  - Light emits from point
- Ammount of "light" constant in a closed environment
- Solve for this steady-state
- Approximate integral

Next Time

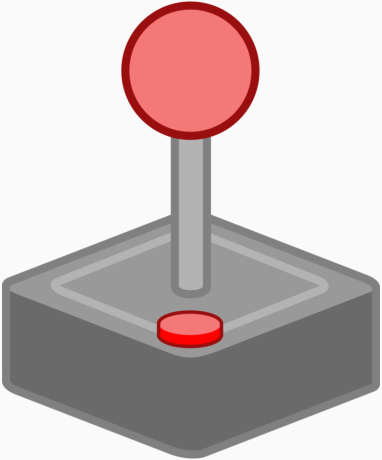
Lecture Global Illumination

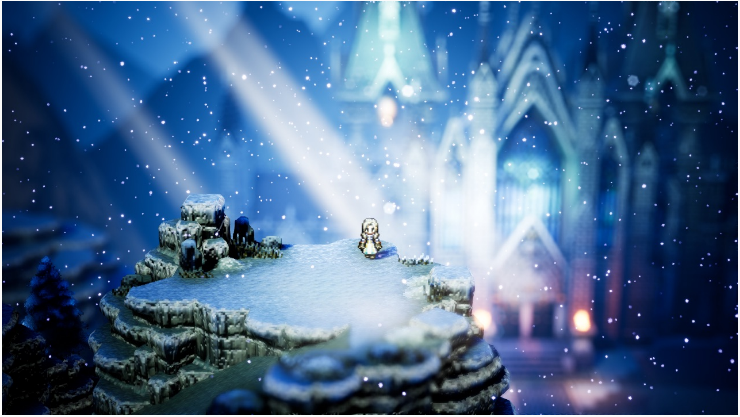
- Introduce concepts
- Formulate problem

Lab Rasterisation & Raytracing

- Try to finish 50% mark of both courseworks this week if you aim for extensions

Homework






Dragon
N

eof