Lecture 16: Rendering Algorithm Overview and Rasterization
Logistics

- PA1 is due 10/30, 11:30 AM.
Graphics

Modeling → Lighting, Camera, and Material → Rendering

http://www.cgarena.com/freestuff/tutorial/max/thomas_highway/sergeant.html
Rendering Algorithm

- Two fundamental problems: visibility and shading
- Visibility: what part of the scene is visible by the camera?
  - For each image pixel, which point in the scene corresponds to it?
- Shading: how does the visible part look like?
  - What’s the color of each image pixel?
- Theoretically shading is independent of visibility, but ray tracing makes some realistic shading (e.g., global illumination) easier to implement

https://www.scratchapixel.com/lessons/3d-basic-rendering/rendering-3d-scene-overview/light-simulator
Visibility Problem

- Two fundamental classes of visibility algorithms
  - Object-centric (Rasterization)
  - Image-centric (Ray tracing)

Rasterization asks: given a point \( P [x, y, z] \), what's the corresponding pixel coordinates \([u, v]\) on the camera sensor?

Ray tracing asks: given a pixel \([u, v]\) on the sensor, what's the associated point in the scene?
Visibility Algorithm

- Rasterization is generally (much) faster than ray tracing.
- Modern GPUs are well-optimized for rasterization, but hardware that supports real-time ray tracing is emerging.
- Ray tracing generates more photorealistic images, because it allows for realistic shading (global illumination).
- RenderMan (REYES) from Pixar is based on rasterization.
  ▶ Considered to be one of the best rasterization algorithm ever to be built
  ▶ Today’s rasterization pipeline has many similarities with REYES
- Pixar now uses RIS, which is purely based on ray tracing.
Once we know which scene point corresponds to which pixel, we need to calculate the pixel color, which is often the point color, but how do we know the color of a point?

Must calculate the light transport that contributes energy to the eyes (camera)
Shading

Once we know which scene point corresponds to which pixel, we need to calculate the pixel color, which is often the point color, but how do we know the color of a point?

- Must calculate the light transport that contributes energy to the eyes (camera)

Accurate shading requires tracing both indirect and direct lights. Also need to the surface material.
Simulate Light Transport

Direct Lighting Only

Direct + Indirect Lighting

Pre-requisite: Unifying the Coordinate Systems

- For convenience and for reusing the same objects across scenes.
- The two killeroos are exactly the same object, but are placed differently in the same scene.
- Define the mesh of the killeroo once with respect to its local coordinate system, and transform it properly when place it in the world coordinate system.
Why Transforming Coordinate Systems?

- A scene description file from *pbrt*, a pretty famous ray tracer.

**Different transformations (translations) when placed in the scene**

- Translate 100 200 -140
  - Include "geometry/killeroo.pbrt"
  - Material "plastic" "color Ks" [.3 .3 .3]
  - "float roughness" [.15]

- Translate -200 0 0
  - Include "geometry/killeroo.pbrt"

Object description in its local coordinate system (not shown here)
Why Transforming Coordinate Systems?

- What is the distance between P1 and P2?
- Need to put them in the same coordinate system first.
- \([1, 3] - [5, 4]\) is meaningless
What Local Coordination Systems Do We Need?

- Objects
  - Point light (shapeless)
  - Area light
  - Distant light
  - Arbitrary shapes

- Light sources
  - Point light (shapeless)
  - Area light
  - Distant light
  - Arbitrary shapes

- Camera
  - A special local frame, where everything else eventually has to be translated to.

Scene

Local frame 1
Local frame 2
Local frame 3
Local frame 4 (light)
Camera frame 1
Camera frame 2
World frame
How to Transform Coordinate Systems?

Local to world transformation (objects and lights)

World to camera transformation

3D to 2D transformation in camera space
Rasterization

Perspective projection: Mapping each triangle to camera sensor/canvas

Rasterization: what pixels on the canvas belong to a particular projected triangle?

Visibility: when multiple points map to the same pixel, which one is visible?
Camera Projection: Where 3D Becomes 2D

- **Fundamental problem**: given a point P [x, y, z], what’s the corresponding pixel coordinates [u, v], if any, on the camera sensor?
  - A point might not been seen by the sensor because of occlusion and/or FOV.
- There are many ways to project a 3D point to a 2D pixel. The most common one is called “perspective projection”
  - Which mimics how human eyes work, which are what many cameras are built to mimic.
  - But there are other projections that you can implement (after all, graphics is just simulation), and many cameras that are built not to mimic human eyes (e.g., fish-eye cameras).
Different Camera Projections

Perspective Projection

Orthographic Projection

Different Camera Projections

Environmental camera

Placing the Image Plane Before the Camera

- We assume the sensor is in front of the pinhole — not possible physically, but simplifies things.
- Scene points could be either before or after the image plane, i.e., does not artificially restrict where a scene point can be.

Figure from “Computer Vision: Models, Learning, and Inference”, Simon J.D. Prince
Perspective Projection

- **Goal**: convert $P \ [x, y, z]$ to pixel coordinates $[u, v]$ on the camera sensor (sensor size $H \times W$ with a focal length $f$).
  - We want to convert by using a transformation matrix.

- **Two steps**:
  - Calculate the $[x', y']$ coordinates of $P'$ in the camera space
  - Convert $[x', y']$ from the camera space to the $[u, v]$ coordinates in the pixel space (most often called the "raster space")
Perspective Projection

\[
\frac{f}{z} = \frac{y'}{y}, \quad \frac{f}{z} = \frac{y'}{y}, \quad \frac{f}{z} = \frac{x'}{x}, \quad \frac{f}{z} = \frac{x'}{x}.
\]

\[P' \text{ is on a 2D plane, so its } z\text{-coordinate isn't important (for now). Let's say } z' = 1.\]

\[z' = 1\]
Perspective Projection Matrix

\[
\begin{bmatrix}
T_{00}, T_{01}, T_{02}, T_{03} \\
T_{10}, T_{11}, T_{12}, T_{13} \\
T_{20}, T_{21}, T_{22}, T_{23} \\
T_{30}, T_{31}, T_{32}, T_{33}
\end{bmatrix}
\]

\[
\begin{bmatrix}
x, y, z, 1 \end{bmatrix}
\times
\begin{bmatrix}
x' \\
y' \\
z' \\
1
\end{bmatrix} = [x', y', z', 1]
\]

\[
x' = xT_{00} + yT_{10} + zT_{20} + T_{30} = xf/z
\]

No \( T_{00}, T_{10}, T_{20}, T_{30} \) would satisfy this universally
Perspective Projection Matrix

\[
\begin{bmatrix}
T_{00}, & T_{01}, & T_{02}, & T_{03} \\
T_{10}, & T_{11}, & T_{12}, & T_{13} \\
T_{20}, & T_{21}, & T_{22}, & T_{23} \\
T_{30}, & T_{31}, & T_{32}, & T_{33}
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= \begin{bmatrix}
x'k \\
y'k \\
z'k \\
k
\end{bmatrix}
\]

Recall:

\[
HC \begin{bmatrix} x'k, & y'k, & z'k, & k \end{bmatrix} \leftrightarrow CC \begin{bmatrix} x', & y', & z' \end{bmatrix}
\]
Perspective Projection Matrix

\[
\begin{bmatrix}
T_{00}, T_{01}, T_{02}, T_{03} \\
T_{10}, T_{11}, T_{12}, T_{13} \\
T_{20}, T_{21}, T_{22}, T_{23} \\
T_{30}, T_{31}, T_{32}, T_{33}
\end{bmatrix}
\]

\[
\begin{bmatrix}
[x, y, z, 1]\end{bmatrix} \times
\begin{bmatrix}
T_{00}, T_{01}, T_{02}, T_{03} \\
T_{10}, T_{11}, T_{12}, T_{13} \\
T_{20}, T_{21}, T_{22}, T_{23} \\
T_{30}, T_{31}, T_{32}, T_{33}
\end{bmatrix}
\]

\[= \begin{bmatrix} [x'k, y'k, z'k, k] \end{bmatrix} \]

Recall: HC \([x'k, y'k, z'k, k] \leftrightarrow CC \[[x', y', z']\]

\[k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z\]

\[
\begin{bmatrix}
0 \\
0 \\
1 \\
0
\end{bmatrix}
\]
Perspective Projection Matrix

\[
[x, y, z, 1] \times \begin{bmatrix}
T_{00}, & T_{01}, & T_{02}, & 0 \\
T_{10}, & T_{11}, & T_{12}, & 0 \\
T_{20}, & T_{21}, & T_{22}, & 1 \\
T_{30}, & T_{31}, & T_{32}, & 0 \\
\end{bmatrix} = [x'k, y'k, z'k, k]
\]

\[
k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z
\]

Recall: HC \([x'k, y'k, z'k, k]) \leftrightarrow CC \([x', y', z']\)
Perspective Projection Matrix

\[
\begin{bmatrix}
T_{00}, T_{01}, T_{02}, 0 \\
T_{10}, T_{11}, T_{12}, 0 \\
T_{20}, T_{21}, T_{22}, 1 \\
T_{30}, T_{31}, T_{32}, 0
\end{bmatrix}
\begin{bmatrix}
x, y, z, 1
\end{bmatrix} =
\begin{bmatrix}
x', y', z', k
\end{bmatrix}
\]

Recall: \( HC [x'k, y'k, z'k, k] \leftrightarrow CC [x', y', z'] \)

\[
k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z
\]

\[
z'k = k = z = xT_{02} + yT_{12} + zT_{22} + T_{32}
\]
Perspective Projection Matrix

\[
\begin{bmatrix}
T_{00}, T_{01}, & 0, & 0 \\
T_{10}, T_{11}, & 0, & 0 \\
T_{20}, T_{21}, & 1, & 1 \\
T_{30}, T_{31}, & 0, & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
= \begin{bmatrix}
x'k \\
y'k \\
z'k \\
k
\end{bmatrix}
\]

Recall:

HC \( [x'k, y'k, z'k, k] \) <=> CC \( [x', y', z'] \)

\[
k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z
\]

\[
z'k = k = xT_{02} + yT_{12} + zT_{22} + T_{32}
\]
Perspective Projection Matrix

\[
\begin{bmatrix}
T_{00}, T_{01}, 0, 0 \\
T_{10}, T_{11}, 0, 0 \\
T_{20}, T_{21}, 1, 1 \\
T_{30}, T_{31}, 0, 0
\end{bmatrix}
\begin{bmatrix}
x, y, z, 1
\end{bmatrix}
= \begin{bmatrix}
x'k, y'k, z'k, k
\end{bmatrix}
\]

Recall:

\[
HC \begin{bmatrix} x'k, y'k, z'k, k \end{bmatrix} \leftrightarrow CC \begin{bmatrix} x', y', z' \end{bmatrix}
\]

\[
k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z
\]

\[
y'k = yf = xT_{01} + yT_{11} + zT_{21} + T_{31}
\]

\[
z'k = k = z = xT_{02} + yT_{12} + zT_{22} + T_{32}
\]
Perspective Projection Matrix

\[
\begin{bmatrix}
T_{00}, & 0, & 0, & 0 \\
T_{10}, & f, & 0, & 0 \\
T_{20}, & 0, & 1, & 1 \\
T_{30}, & 0, & 0, & 0
\end{bmatrix}
\begin{bmatrix}
x, \\
y, \\
z, \\
1
\end{bmatrix} = \begin{bmatrix}
x'k, \\
y'k, \\
z'k, \\
k
\end{bmatrix}
\]

Recall: HC \([x'k, y'k, z'k, k]\) \(\Leftrightarrow\) CC \([x', y', z']\)

\[
x' = \frac{x}{z} \\
y' = \frac{y}{z} \\
z' = 1
\]

\[
k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z
\]

\[
y'k = yf = xT_{01} + yT_{11} + zT_{21} + T_{31}
\]

\[
z'k = k = xT_{02} + yT_{12} + zT_{22} + T_{32}
\]
Perspective Projection Matrix

\[
\begin{bmatrix}
  T_{00}, & 0, & 0, & 0 \\
  T_{10}, & f, & 0, & 0 \\
  T_{20}, & 0, & 1, & 1 \\
  T_{30}, & 0, & 0, & 0
\end{bmatrix}
\]

\[
[x, y, z, 1] \times \begin{bmatrix}
  T_{00}, & 0, & 0, & 0 \\
  T_{10}, & f, & 0, & 0 \\
  T_{20}, & 0, & 1, & 1 \\
  T_{30}, & 0, & 0, & 0
\end{bmatrix} = [x'k, y'k, z'k, k]
\]

Recall: HC \([x'k, y'k, z'k, k] \Leftrightarrow CC [x', y', z']\)

\[
x' = \frac{x}{z} \quad y' = \frac{y}{z} \quad z' = 1
\]

\[
k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z
\]

\[
y'k = yf = xT_{01} + yT_{11} + zT_{21} + T_{31}
\]

\[
x'k = xf = xT_{00} + yT_{10} + zT_{20} + T_{30}
\]

\[
z'k = k = z = xT_{02} + yT_{12} + zT_{22} + T_{32}
\]
Perspective Projection Matrix

\[
[x, y, z, 1] \times \begin{bmatrix}
f, 0, 0, 0 \\
0, f, 0, 0 \\
0, 0, 1, 1 \\
0, 0, 0, 0 \\
\end{bmatrix} = [x'k, y'k, z'k, k]
\]

Recall: \HC [x'k, y'k, z'k, k] <=> CC [x’, y’, z’]

\[
k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z
\]

\[
x'k = xf = xT_{00} + yT_{10} + zT_{20} + T_{30}
\]

\[
y'k = yf = xT_{01} + yT_{11} + zT_{21} + T_{31}
\]

\[
z'k = k = z = xT_{02} + yT_{12} + zT_{22} + T_{32}
\]
Perspective Projection Matrix

\[
\begin{bmatrix}
    f & 0 & 0 & 0 \\
    0 & f & 0 & 0 \\
    0 & 0 & 1 & 1 \\
    0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
[x, y, z, 1] \times \begin{bmatrix}
    f & 0 & 0 & 0 \\
    0 & f & 0 & 0 \\
    0 & 0 & 1 & 1 \\
    0 & 0 & 0 & 0
\end{bmatrix} = [x'k, y'k, z'k, k]
\]

Recall:
HC \[x'k, y'k, z'k, k\] \(\Leftrightarrow\) CC \([x', y', z']\)

\[
x' = \frac{x}{z} \quad y' = \frac{y}{z} \quad z' = 1
\]

\[
k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z
\]

\[
x'k = xf = xT_{00} + yT_{10} + zT_{20} + T_{30}
\]

\[
y'k = yf = xT_{01} + yT_{11} + zT_{21} + T_{31}
\]

\[
z'k = k = xT_{02} + yT_{12} + zT_{22} + T_{32}
\]
Perspective Projection Matrix

\[
\begin{bmatrix}
  f & 0 & 0 & 0 \\
 0 & f & 0 & 0 \\
 0 & 0 & 1 & 1 \\
 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[ [x, y, z, 1] \times
\begin{bmatrix}
  f & 0 & 0 & 0 \\
 0 & f & 0 & 0 \\
 0 & 0 & 1 & 1 \\
 0 & 0 & 0 & 0
\end{bmatrix}
= [x', y', z', k]
\]

\[ [x, y, z, 1] \rightarrow [xf/z, yf/z, 1]
\]

\[ k = xT_{03} + yT_{13} + zT_{23} + T_{33} = z
\]

\[ x'k = xf = xT_{00} + yT_{10} + zT_{20} + T_{30}
\]

\[ y'k = yf = xT_{01} + yT_{11} + zT_{21} + T_{31}
\]

Recall: \( HC [x'k, y'k, z'k, k] \Leftrightarrow CC [x', y', z'] \)

Added after the lecture: why does \( T_{23} \) have to be 1? Because this is the only way we can make the matrix universal. Look at the expression of \( x' \) and \( y' \); they both have \( z \) as the denominator. Meanwhile we need to divide by \( k \) when converting \( HC \) to \( CC \). So if \( k \) is \( z \), then the matrix could be made universal.
Perspective Projection Matrix: Mind the Z-Axis

- Our matrix so far will always translate z-coordinate of any P to the same $z' = 1$. Good?
- P1 and P2 are projected to the same point $P'$, but P1 is visible and P2 is not. Critical for a rendering engine to know.
- Somehow we need to make sure $z_1' < z_2'$ after projection.
Perspective Projection Matrix: Mind the Z-Axis

\[ x' = \frac{x}{z} f \quad y' = \frac{y}{z} f \quad z' = z \]

\[
\begin{bmatrix}
    f & 0 & T_{02} & 0 \\
    0 & f & T_{12} & 0 \\
    0 & 0 & T_{22} & 1 \\
    0 & 0 & T_{32} & 0 \\
\end{bmatrix}
\]

- Try 1: keep z the same before and after transformation
- Problem: No one single matrix that universally works for all possible z values

\[ z'k = zk = z^2 = xT_{00} + yT_{10} + zT_{20} + T_{30} \]
Perspective Projection Matrix: Mind the Z-Axis

\[ x' = \frac{x}{z} f \quad y' = \frac{y}{z} f \quad z' = Cz \]

\[
\begin{bmatrix}
  f & 0 & T_{02} & 0 \\
  0 & f & T_{12} & 0 \\
  0 & 0 & T_{22} & 1 \\
  0 & 0 & T_{32} & 0
\end{bmatrix}
\]

- Try 2: scale z with a constant, say C.
- Same problem as before.
- We need to \textit{bound} z.

\[ z'k = Czk = Cz^2 = xT_{00} + yT_{10} + zT_{20} + T_{30} \]
Try 3: project the smallest z to 0 and largest z to 1.

Need to have an artificial smallest z value and largest z value.

- Effectively, there is an artificial “near clipping plane” $N$ and an artificial “far clipping” plane $F$.
- Only points between these two planes are visible to the camera — purely artificial just to get our algorithm work, no real physical meaning.
- Image plane can be anywhere. It’s not related to $N$ and $F$ (OpenGL assumes $N == f$).
Perspective Projection Matrix: Mind the Z-Axis

- Try 3: project the smallest $z$ to 0 and largest $z$ to 1.
- Need to have an artificial smallest $z$ value and largest $z$ value.
  - Effectively, there is an *artificial* “near clipping plane” $N$ and an *artificial* “far clipping” plane $F$.
  - Only points between these two planes are visible to the camera — purely artificial just to get our algorithm work, no real physical meaning.
  - Image plane can be anywhere. It’s not related to $N$ and $F$ (OpenGL assumes $N == f$).
Try 3: project the smallest $z$ to 0 and largest $z$ to 1.

Need to have an artificial smallest $z$ value and largest $z$ value.

- Effectively, there is an *artificial* “near clipping plane” $N$ and an *artificial* “far clipping” plane $F$.
- Only points between these two planes are visible to the camera — purely artificial just to get our algorithm work, no real physical meaning.
- Image plane can be anywhere. It’s not related to $N$ and $F$ (OpenGL assumes $N == f$).
Perspective Projection Matrix: Mind the Z-Axis

\[
\begin{bmatrix}
    x, y, z, 1
\end{bmatrix}
\times
\begin{bmatrix}
    f & 0 & T_{02} & 0 \\
    0 & f & T_{12} & 0 \\
    0 & 0 & T_{22} & 1 \\
    0 & 0 & T_{32} & 0 \\
\end{bmatrix}
= \begin{bmatrix}
    x'k, y'k, z'k, k
\end{bmatrix}
\]

\[z'k = xT_{02} + yT_{12} + zT_{22} + T_{32}\]
Perspective Projection Matrix: Mind the Z-Axis

\[
\begin{bmatrix}
\mathbf{x}'k, \mathbf{y}'k, \mathbf{z}'k, k
\end{bmatrix} = \begin{bmatrix}
f & 0 & 0 & 0 \\
0 & f & 0 & 0 \\
0 & 0 & T_{22} & 1 \\
0 & 0 & T_{32} & 0 \\
\end{bmatrix} \begin{bmatrix}
x, y, z, 1
\end{bmatrix}
\]

\[z'k = xT_{02} + yT_{12} + zT_{22} + T_{32}\]
Perspective Projection Matrix: Mind the Z-Axis

\[
\begin{bmatrix}
x, y, z, 1
\end{bmatrix} \times \begin{bmatrix}
f & 0 & 0 & 0 \\
0 & f & 0 & 0 \\
0 & 0 & T_{22} & 1 \\
0 & 0 & T_{32} & 0
\end{bmatrix} = \begin{bmatrix}
x'k, y'k, z'k, k
\end{bmatrix}
\]

\[z'k = xT_{02} + yT_{12} + zT_{22} + T_{32}\]

\[N \times T_{22} + T_{32} = 0\]

\[F \times T_{22} + T_{32} = 1\]
Perspective Projection Matrix: Mind the Z-Axis

\[
\begin{bmatrix}
 f & 0 & 0 & 0 \\
 0 & f & 0 & 0 \\
 0 & 0 & T_{22} & 1 \\
 0 & 0 & T_{32} & 0
\end{bmatrix}
\begin{bmatrix}
 x \\
 y \\
 z \\
 1
\end{bmatrix}
= \begin{bmatrix}
 x'k \\
 y'k \\
 z'k \\
 k
\end{bmatrix}
\]

\[z'k = xT_{02} + yT_{12} + zT_{22} + T_{32}\]

\[NT_{22} + T_{32} = 0 \quad \Rightarrow \quad T_{22} = -F/(N-F)\]

\[FT_{22} + T_{32} = 1 \quad \Rightarrow \quad T_{32} = FN/(N-F)\]
Perspective Projection Matrix: Mind the Z-Axis

\[
\begin{bmatrix}
  f & 0 & 0 & 0 \\
  0 & f & 0 & 0 \\
  0 & 0 & \frac{F}{(F-N)} & 1 \\
  0 & 0 & \frac{FN}{(N-F)} & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= \begin{bmatrix}
x' \\
y' \\
z' \\
k
\end{bmatrix}
\]

\[z'k = xT_{02} + yT_{12} + zT_{22} + T_{32}\]

\[\begin{align*}
NT_{22} + T_{32} &= 0 \\
FT_{22} + T_{32} &= 1
\end{align*}\]

\[T_{22} = -\frac{F}{(N-F)} \quad T_{32} = \frac{FN}{(N-F)}\]
Perspective Projection Matrix: Mind the Z-Axis

\[
\begin{bmatrix}
    x', y', z', 1 \\
\end{bmatrix} = \begin{bmatrix}
    f & 0 & 0 & 0 \\
    0 & f & 0 & 0 \\
    0 & 0 & \frac{F}{(F-N)} & 1 \\
    0 & 0 & \frac{FN}{(N-F)} & 0 \\
\end{bmatrix} \begin{bmatrix}
    x, y, z, 1 \\
\end{bmatrix}
\]

\[
z'k = xT_{02} + yT_{12} + zT_{22} + T_{32}
\]

\[
\begin{align*}
NT_{22} + T_{32} &= 0 \\
FT_{22} + T_{32} &= 1
\end{align*}
\]

\[
x' = \frac{x}{z}f \\
y' = \frac{y}{z}f \\
z' = \frac{F(N - z)}{z(N - F)}
\]
Near/Far Clipping Plane

- N and F are independent of focal length (f).
  ▶ After all, f is real while N and F are artificial
- E.g., N could be before f, the same transformation still holds.

\[
\begin{align*}
x' &= \frac{x}{f} \\
y' &= \frac{y}{f} \\
z' &= \frac{F(N - z)}{z(N - F)}
\end{align*}
\]

\[
\begin{align*}
x' &= \frac{x}{z} \\
y' &= \frac{y}{z} \\
z' &= \frac{F(N - z)}{z(N - F)}
\end{align*}
\]
Perspective Projection/Transformation Matrix

- Perspective projection is not an affine transformation.
- Perspective projection mimics a pinhole camera model.
- Perspective projection is not needed in/used by ray tracing.
Generating Pixel Coordinates

- Where we are now: we could generate \([x', y']\) (ignore \(z'\) when discussing one single point \(P\)) in the **camera space** for any given scene point \([x, y, z]\).
- Next: generating the \([u, v]\) pixel coordinates on the sensor.
Generating Pixel Coordinates

\[ P \ [x, y, z] \]

\[ P' \ [x', y'] \]

(Virtual) image plane/sensor

Camera/Eye (Origin of the camera space)

\[ [u, v] \]

Image Plane/Canvas/Screen Space

\[ [-W/2, -H/2] \]
\[ [W/2, -H/2] \]

\[ [W/2, H/2] \]
\[ [-W/2, H/2] \]

Focal length \( f \)
Generating Pixel Coordinates

Image Plane/Canvas/Screen Space

Raster Space (PW pixels wide and PH pixels high)

$$u = \left\lfloor \frac{x' + W/2}{W} \times PW \right\rfloor$$

$$v = \left\lfloor \frac{y' - H/2}{H} \times PH \right\rfloor$$
Boundary Test

P can’t be captured by the camera if $|x'| > W/2$ or $|y'| > H/2$
Rasterization

- Once we project the three vertices to the pixels, the next question is to figure out what pixels are within the triangle. After all, we render triangles not pixels.

- Naively, loop over all pixels in the image, and test whether a pixel is inside the triangle. We could prune the search space using the triangle’s bounding box.

- The key challenge: how do we know if a pixel is inside a triangle?

```plaintext
Foreach triangle in mesh
  Perspective project triangle to canvas;
  Foreach pixel in image
    if (pixel is in the projected triangle)
      pixel.color = generateColor();
```
Pixel-in-Triangle Test Using Barycentric Coordinates

\[ V = \lambda_1 A + \lambda_2 B + \lambda_3 C \]
\[ \lambda_1 + \lambda_2 + \lambda_3 = 1 \]

\[ \lambda_1 = \text{Area}(AVB)/\text{Area}(ABC) \]
\[ \lambda_2 = \text{Area}(BVC)/\text{Area}(ABC) \]
\[ \lambda_3 = \text{Area}(CVA)/\text{Area}(ABC) \]

https://en.wikipedia.org/wiki/Barycentric_coordinate_system
Pixel-in-Triangle Test Using Barycentric Coordinates

For any $V$ that's inside the triangle:

$$0 \leq \lambda_1, \lambda_2, \lambda_3 \leq 1$$

For any $V$ that's outside the triangle:

Any of $\lambda_1, \lambda_2, \lambda_3$ is outside the $[0, 1]$ range.

$$V = \lambda_1 A + \lambda_2 B + \lambda_3 C$$

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-stage
When multiple points in the scene get projected to the same pixel, must determine which point “wins”, i.e., gets to assign its color to the pixel. This is called the visibility problem or hidden surface problem.

Fortunately, perspective projection maintains the relative point depth. Determining the relative depth is done using a depth-buffer or a z-buffer.

Foreach triangle in mesh
Perspective project triangle to canvas;
Foreach pixel in image
  if (pixel is in the projected triangle)
    D = computeDepth(pixel)
    if (D < depthBuffer[pixel])
      shade(pixel.color)
      depthBuffer[pixel] = D
Calculating Depth for Non-Vertex Pixels

- Given a pixel, how do we recover its z-coordinate value in the camera space?
- We need the z-coordinate to compare depths of different points.
- What we have: perspective projection converts z-coordinate from the camera space to the image space, but only for the vertices.

![Diagram showing perspective projection and clipping planes](https://www.scratchapixel.com)

© www.scratchapixel.com
Calculating Depth for Non-Vertex Pixels

- For pixels that correspond to vertices, reverse the perspective projection will get us their z-coordinates in the camera space. Done!
- For pixels that correspond to other points on a triangle, interpolate their depths using the barycentric coordinates, but in the camera space!
- Be careful: directly interpolating in the sensor plane is incorrect!

\[
V = \lambda_1 A + \lambda_2 B + \lambda_3 C \\
\lambda_1 + \lambda_2 + \lambda_3 = 1
\]
Reference Material

- Barycentric coordinates:
  - Deriving the expression of barycentric coordinates
  - Code for calculating it given the three vertices
- Rasterization: a Practical Implementation. A relatively detailed description of the three rasterization steps with code.
- A Simple Perspective Matrix.