

Computer Graphic Note

Jiechang Shi

Version: 1.00

Update: September 25, 2019

1 Class 1: Overview

1. Frame Buffer

- Pixel: One element of a frame buffer
- Pixel depth: Number of bytes per-pixel in the buffer
- Resolution: Width \times Height
- Buffer size: Total memory allocated for frame buffer
- **Exam Question:** Given Resolution and Pixel depth, Asked Buffer size
- Z Value: For solving the hidden-surface removal(HSR) problem

2 Class 2,3: Rasterization

1. For each pixel on screen, the sample point of that pixel is the center point of that pixel, which have **integer** coordinates. For examples, (3, 2).
2. A simple problem: Rasterizing Lines

Program Description: Given two endpoints, $P = (x_0, y_0)$, $R = (x_1, y_1)$ find the pixels that make up the line.

Note that: Lines are infinitely thin so they rarely fall on pixel sample point.

A Feasible Description: Rasterize lines as **closest** pixels to actual lines, with 2 requirement

- No Gap
- Minimize error(distance to true line)

To make this question simplify: Only consider situation that $|x_1 - x_0| \geq |y_1 - y_0| \geq 0 \wedge |x_1 - x_0| \neq 0$, which means the $-1 \leq slope \leq 1$. Otherwise we just exchange x and y .

A basic Algorithm: $k = \frac{y_1 - y_0}{x_1 - x_0}$, $d = y_0 - kx_0$, for each $x_0 \leq x \leq x_1$, $y = ROUND(kx + d)$. This method by brute force is inefficient because of the multiplication and the function $ROUND()$.

Basic Incremental Algorithm: for each $x_0 \leq x_i < x_{i+1} \leq x_1$, $y_{i+1} = y_i + k$, However, the successive addition of a real number can lead to a **cumulative error buildup!**

Midpoint Line Algorithm:

- For $0 \leq k \leq 1$
- For one approximate point $P = (x, y)$, we only have 2 choices for the next point $E = (x + 1, y)$ and $NE = (x + 1, y + 1)$, we should choose the one which is closer to $k(x + 1) + d$
- Calculate the middle point $M = (x + 1, y + \frac{1}{2})$

- If the **Intersection point** Q is below M , take E as next, otherwise take NE as next.
- Note that: we consider this equation:

$$f(x, y) = ax + by + c = (y_1 - y_0)x - (x_1 - x_0)y + (x_1y_0 - y_1x_0)$$

We assume $a > 0$

- For a point (x, y)
 - if $f(x, y) = 0$, (x, y) lies on the line.
 - if $f(x, y) < 0$, (x, y) lies upon the line.
 - if $f(x, y) > 0$, (x, y) lies below the line.
- So we have to test $f(M) = a(x + 1) + b(y + \frac{1}{2}) + c = f(Former) + a + \frac{b}{2}$
- Assume $a > 0$, if $f(M) > 0$ choose NE otherwise choose E
- Update $f(Former)$:
 - If we choose E , $f(Former) = f(Former) + a$
 - If we choose NE , $f(Former) = f(Former) + a + b$
- Note that: a and b are **constant integer**, so here is no **cumulative error issue**

3. A harder problem: Triangles Rasterization

Why Triangle:

- Triangles (*tris*) are a simple explicit 3D surface representation.
- Convex and concave polygons (*polys*) can be decomposed into triangles.
- Tris are planar and unambiguously defined by three vertex(*verts*) coordinates (*coords*).

Definition: Find and draw **pixel** samples **inside** *tri* edges and interpolate parameters defined at *verts*

4. Rasterization and Hidden Surface Removal(HSR) Algorithm Classes:

- Image order rasterization: ray tracing/ ray casting
 - traverse pixel, process each in world-space
 - transform rays from image-space to world-space
- Object order rasterization: scan-line / LEE
 - traverse triangles, process each in image-space
 - transform objects from model-space to image-space

5. LEE Linear Expression Evaluation Algorithm:

- We already discussed in *Midpoint Line Algorithm* that how to determine a point is on the left(up) or right(below) the line, just a *quick review here*:
 - Assume the line have a positive *slope*
 - For an Edge Equation E , for point (x, y) : $E(x, y) = dY(x - X) - dX(y - Y)$
 - if $E(x, y) = 0$, (x, y) lies on the line.
 - if $E(x, y) < 0$, (x, y) lies right(below) the line.
 - if $E(x, y) > 0$, (x, y) lies left(up) the line.
- For **Rasterization**:
 - Compute LEE result for all three edges.
 - Pixels with **consistent sign** for all three edges are inside the *tri*.
 - Include **edge pixels** on left or right edges.
- LEE need to check every pixel in the bounding box.
- LEE is very good in parallel(SIMD) system.
- Furthermore: Given 3 random *verts* how to find CW edge cycle
 - Determine L/R and Top/Bot edges for edge-pixel ownership

6. Scan Line Rasterizer:

- Sort vets by Y
 - Setup edge DDAs for edges
 - Sort edges by L or R(The long edge on left or right)
 - Start from Top Vertice, and switch DDA when hit the middle vertice.
7. **Interpolate Z:**
 A general 3D plane equation has 4 terms: $Ax + By + Cz + D = 0$
 (A, B, C) is the normal of that plane, so $(X, Y, Z)_0 \times (X, Y, Z)_1 = (A, B, C)$
 Then plug any vertex coord into equation and solve for D .
 Given (A, B, C, D) and any point (x, y) can solve z
8. **Used Z-buffer to remove hidden surfaces**
 Initial Z-buffer to MAXINT at the begining of every frame
 Interpolate vertex Z values to get Z_{pix}
 Only write new pixel to the buffer if $Z_{pix} < Z_{buffer}$
 Notice that Z should always **bigger or equal to zero!**
9. **Hidden Line Removal(HLR):**
 Simple z-buffer does not work when the render only draws edge(outlines of polygons).
 Need edge-crossing and object sorting methods.
10. **Painter's Algorithm:** render in order front to back
 A object is in front of another object means:
 Z of all verts of one object is less than the other.
 This algorithm not work if Z -sort is ambiguous.
11. **Warnock Algorithm:**
 Subdivide screen untril a leaf region has a simple front/back relationship.
 Leaf regions have one or zero surfaces visible, and the smallest region is usually a pixel
 Usually use quad tree subdivision
12. **BSP-Tree:**
 View-Independent binary tree(pre-calculated) allows a view-dependent front-to-back or back-to-front traversal of surfaces.
 Use Painter Algorithm to do back-to-front traversal.
 Useful for **transparency** - full depth-sort of all surface.
13. **Culling:**
- Culling with portals: pre-compute the invisible part.
 - Culling by View Frustum: Skip a triangle iff all its vertices are beyond **the same screen edge!**
Pitfall: If the vertices are beyond different edge, some part of the *tri* might still in the screen. Image a giant *tri* that cover the whole screen.
 - Backface Culling: For **closed(water-tight)** objects, surfaces with **oriented-normals facing away** from the camera are never visible.
Pitfall: BF Culling only work for water-tight object!
 - Frustum: Only visible triangles are drawn into the frame buffer.

3 Class 4,5,6,7:Transformations

- Linear transformations ($Xforms$) define a mapping of coordinates ($coords$) in one coordinate frame to another.

$$V_b = X_{ba}V_a$$

- **Homogeneous Vector** (V) is 4×1 columns $(x, y, z, w)^T$
- **Homogeneous Transforms** (X) is 4×4 matrix
- From Homogeneous Vector to 3D Vector:

$$x = \frac{x}{w}, y = \frac{y}{w}, z = \frac{z}{w}$$

- **Why** we use Homogeneous Vector?

We want to uniform the transform matrix including translation, scaling, rotation in the same form of matrix.

3.1 Transformation Matrix

1. **Translation:**

$$T(t_x, t_y, t_z) \Rightarrow \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, T^{-1}(t_x, t_y, t_z) = T(-t_x, -t_y, -t_z)$$

2. **Scaling:**

$$S(s_x, s_y, s_z) \Rightarrow \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, S^{-1}(s_x, s_y, s_z) = S\left(\frac{1}{s_x}, \frac{1}{s_y}, \frac{1}{s_z}\right)$$

3. **Rotation, CCW:**

$$R_x(\theta) \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_x^{-1}(\theta) = R_x^T(\theta)$$

$$R_y(\theta) \Rightarrow \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_y^{-1}(\theta) = R_y^T(\theta)$$

$$R_z(\theta) \Rightarrow \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, R_z^{-1}(\theta) = R_z^T(\theta)$$

4. Pitfall: commutative property is for S,R only.

Here assume **uniform scaling** in all dimensions.

If S is not an uniform scaling matrix. S,R don't have commutative property.

3.2 Spaces Transformation

1. **NDC to Output Device**

$$X_{sp} \Rightarrow \begin{bmatrix} \frac{xs}{2} & 0 & 0 & \frac{xs}{2} \\ 0 & -\frac{ys}{2} & 0 & \frac{ys}{2} \\ 0 & 0 & MAXINT & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that:

Output Device is **RH coords** and origin in **upper left**. $X \in [0, xs), Y \in [0, ys), Z \in [0, MAXINT]$

NDC is **LH coords** and origin at screen center. $X, Y \in [-1, 1], Z \in [0, 1]$

2. Perspective Projection

$$X_{pi} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \\ 0 & 0 & \frac{1}{d} & 1 \end{bmatrix}$$

What is d And Why there are two $\frac{1}{d}$?

- Assume camera is on $(0, 0, -d)$, perspective(also image) plane is $z = 0$, a object in world space is (X, Y, Z)
- Defined FOV(field of view) as the angle the camera can see.
- Note: $X \in [-1, 1]$, so the distant(d) from Forcus point to view plane can be calculate by this equation:

$$\frac{1}{d} = \tan\left(\frac{FOV}{2}\right)$$

- Futher More: The object project to view plane can be calculate by these equations:

$$\frac{X}{Z+d} = \frac{x}{d} \Rightarrow x = \frac{X}{\frac{Z}{d}+1}$$

$$\frac{Y}{Z+d} = \frac{y}{d} \Rightarrow y = \frac{Y}{\frac{Z}{d}+1}$$

$$\frac{Z}{Z+d} = \frac{z}{d} \Rightarrow z = \frac{Z}{\frac{Z}{d}+1}$$

$$(x, y, z) = \left(\frac{X}{\frac{Z}{d}+1}, \frac{Y}{\frac{Z}{d}+1}, \frac{Z}{\frac{Z}{d}+1}\right)$$

- Write this 3D vector to Homogeneous Vector:

$$(x, y, z, w) = \left(X, Y, Z, \frac{Z}{d}+1\right)$$

- Futher: We focus on the **range** of Z now is $z \in (-\infty, d)$.

But in NDC we hope $z \in [0, 1)$ So:

We delete all vector that $Z < 0$, because they cannot project to the view plane.

For $z \geq 0$, we define $z' = \frac{z}{d}$

- $(x, y, z', w) = \left(X, Y, \frac{Z}{d}, \frac{Z}{d}+1\right) = X_{pi} * (X, Y, Z, 1)$

Pitfall: Do Z interpolation in Perspective Plane!

Why we need the fareset plane?

Asymptotic curve of Z vs. z , that z increase slower when Z is large.

It might map different Z to the same z

3.3 Camera Matrix

- Assume camera position is c , camera look-at point is l , here c and l are both in world coordinate. And the world up vector is \vec{up}
- Camera Z-axis **in world coordinate** is

$$\vec{Z} = \frac{\vec{cl}}{\|\vec{cl}\|}$$

3. Camera Y-axis **in world coordinate** is the orthogonal(vertical) part of world-up vector to Z-axis which is

$$\vec{up'} = \vec{up} - (\vec{up} \cdot \vec{Z})\vec{Z}$$

$$\vec{Y} = \frac{\vec{up'}}{\|\vec{up'}\|}$$

4. Camera X-axis **in world coordinate** is orthogonal to both Y and Z axes. So:

$$\vec{X} = \vec{Y} \times \vec{Z}$$

5. Build the X_{wi} from camera space to world space:

X-axis vector [1,0,0] in camera space should be \vec{X} in world space, also for Y,Z-axis vectors, So:

$$X_{wi} \Rightarrow \begin{bmatrix} X_x & Y_x & Z_x & 0 \\ X_y & Y_y & Z_y & 0 \\ X_z & Y_z & Z_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6. Also we need to add the translation of the camera to the Matrix:

$$X_{wi} \Rightarrow \begin{bmatrix} X_x & Y_x & Z_x & c_x \\ X_y & Y_y & Z_y & c_y \\ X_z & Y_z & Z_z & c_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

7. Now we can get the inverst matrix X_{iw} :

$$X_{iw} \Rightarrow \begin{bmatrix} X_x & X_y & X_z & -X \cdot c \\ Y_x & Y_y & Y_z & -Y \cdot c \\ Z_x & Z_y & Z_z & -Z \cdot c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

8. In this proof we know that: If we know the X,Y,Z-axis in world coordinate for a specific space, we can easily build and inverst the translation from or to that space.

This method can also be used to **proof the general rotation matrix**.

9. Orbit a Model about a Point

The idea is the same as place camera.

Need to care about **which space** you current in!