I. Sampling & Aliasing

1. Nyquist - Shannon Theorem.

For a band-limited (with no frequencies above threshold w_0) signal, the signal can be perfectly reconstructed if sampled with $T = \frac{1}{2} w_0$.

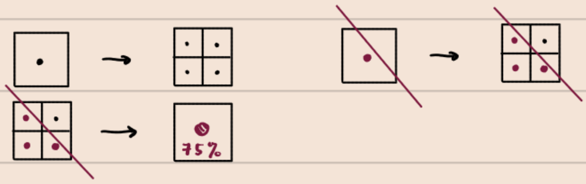
2. In practical, sampling is imperfect, so artifacts occur.

- Jaggies - Moiré Pattern - Wagon Wheel Illusion.

3. Super sampling

① Increase frequency of sampling

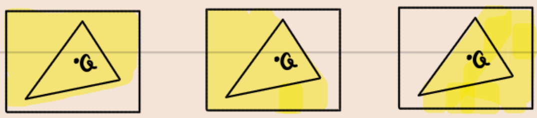
② Resample to display's pixel rate.



II. Point-in-triangle Test.

1. Test if a point Q is inside a triangle $\triangle P_1P_2P_3$?

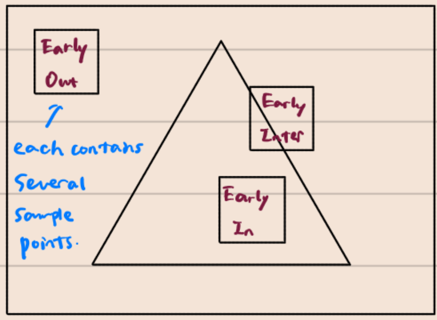
① Half-plane Test : see is Q is contained in three half planes.



② Cross product : Essentially the same as above.

$Q \times \overrightarrow{P_0P_1}$, $Q \times \overrightarrow{P_1P_2}$, $Q \times \overrightarrow{P_2P_0}$ see if they are on the "same side".

③ Real Approach : Hardware works this way.



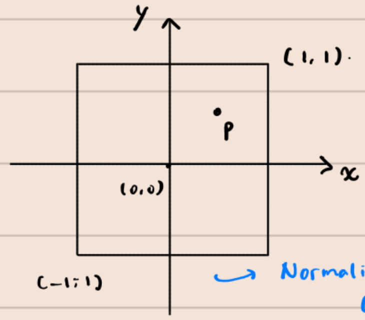
- Check if large blocks intersect the triangle.
  - if not, skip this block entirely. (Early out Case.)
  - if yes,
    - Contained, all the samples in this block is covered. (Early In Case)
    - Intersected, test each sample points in the block (in parallel). (Early Inter Case)

III. Screen Transformation.

After we transforms the geometry from the model space to the canonical $[-1, 1]^3$ cube, we know what to do to map it onto the screen.

Step 1 Drop z. (actually, we say take points from $[-1, 1]^2$ on $z = 1$ plane).

Step 2 Scale to $[0, W] \times [0, H]$.



→ Normalized Device Coordinates (NDC).

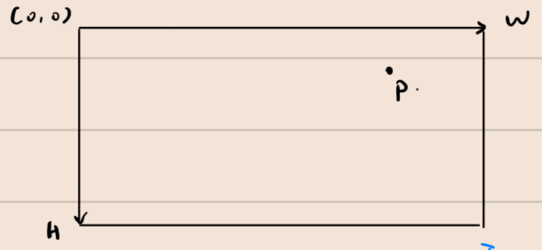


Image Space.

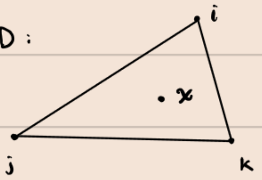
reflect about x → translate by $(1, 1)$ → Scale by $(W/2, H/2)$.

IV. Interpolations.

1. Linear Interpolations.

1D : $\hat{f}(t) = (1-t) f_i + t f_j$.

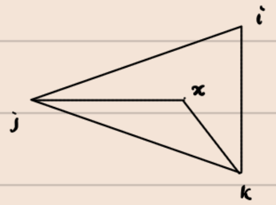
2D :



$f(x) = f(i) \cdot \varphi_i + f(j) \cdot \varphi_j + f(k) \cdot \varphi_k$.

where φ is the barycentric coordinates

2. Barycentric Coordinate.



$\varphi_i(x) = Area(\Delta xjk) / Area(\Delta ijk)$

i.e. $\varphi_j(x) = Area(\Delta xik) / Area(\Delta ijk)$

$\varphi_k(x) = Area(\Delta xij) / Area(\Delta ijk)$

V. Appetizer : Drawing Lines.

1. Midpoint Algorithm.

// draw(x,y) shades the pixel (x,y).

// Requires : $x_1 \geq x_0$. (otherwise flip them).

// Starting @ (x_0, y_0) and ends at (x_1, y_1). f is the line function.

DrawLine(x,y) :

Let $y = y_0$; $d = f(x_0+1, y_0+0.5)$.

For $x = x_0$ to x_1 do :

Draw (x,y).

If $d < 0$ then $y = y+1$; $d = d + (x_1 - x_0) + (y_0 - y_1)$.

Else $d = d + (y_0 - y_1)$.

VI. Anti-Aliasing.

1. Filtering.

Removing certain sampling results is called <u>filtering</u>.

- Removing low-freq : Edges on the image.
- Removing hi-freq : Blurs.
- Removing hi & low-freq : Inner contours.

2. Convolution.

① Kernel : Defines how to do the weighted averaging.

Ex.

Signal	1	3	5	7	3	1	3	8	6	4
Kernel	[1/4	1/2	1/4]							
Result	_	3	_	_	_	_	_	_	_	_

② Box Blur : Kernel is an n×n box, each cell having the same weight. $[-r,r]^2$.

We say the filter function is $f_{box,r}(x) = \begin{cases} \frac{1}{2r} & -r \le x \le r. \\ 0 & o.w. \end{cases}$

③ Gaussian Blur : $f_{g,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2}$.

3. MSAA (Antialiasing by Supersampling).

- As mentioned in I.
- Very expensive.

4. FXAA (Fast Approximate).

- Postprocessing of image.
- Very fast.

5. TAA (Temporal).

- Reuse the sample from last frame.

6. DLSS (Deep Learning Super Sampling).

- "Guess" the colors in the aliased region through deep learning.

VII. Depth.

After all. we still need to care about the depth - the z value.

1. z-buffer.

① Stores the closest triangle seen so far.

(Initialize with ∞).

② Occlusion test is based on depth, so handles intersection well.

(Requires depth information at each sample point).

2. Transparency

① Alpha channel = transparency

② "Over" operator : A over B $\neq$ B over A

Notice that. different from intuition. Over is NOT commutative.

③ Composite Methods.

- Non-premultiplied : $C = \alpha_B B + (1-\alpha_B)\alpha_A A$. $\alpha_C = \alpha_B + (1-\alpha_B)\alpha_A$.
- Premultiplied : $A' = (\alpha_A A_r, \alpha_A A_g, \alpha_A A_b, \alpha_A)$.

$B' = (\alpha_B B_r, \alpha_B B_g, \alpha_B B_b, \alpha_B)$

$\mapsto C' = B' + (1-\alpha_B) A'$. $\Rightarrow C = (C_r, C_g, C_b, \alpha_C) \sim (C_r/\alpha_C, C_g/\alpha_C, C_b/\alpha_C)$.

3. Drawing semi-transparency. Triangles must be rendered in back to front order.

Define over(c1, c2) = c1.rgba + (1-c1.a) * c2.rgba ;

Define UpdateColorBuffer(x, y, sample_color, sample_depth) :

if (PassDepthTest(Sample_depth, zbuffer[x][y]) :

Color[x][y] = over(sample_color, Color[x][y]).

VIII. Pipeline.

Step 1 Getting inputs from model files. (.obj/.fbx/...). CPU.

Step 2 Transforms from obj space to camera space. (M). GPU. Vertex Shader.

Step 3 Transforms from camera space to Normalized Device Coordinates. Space. (VP).

Step 4 Clipping and culling.

Step 5 Transforms to screen coordinates $[0, W] \times [0, H]$.

Step 6 Set up triangles. Sampling coverage. GPU. Fragment Shader.

Step 7 Compute triangle color at sample point.

Step 8 Perform depth/stencil test if enabled. Update color buffer if needed.