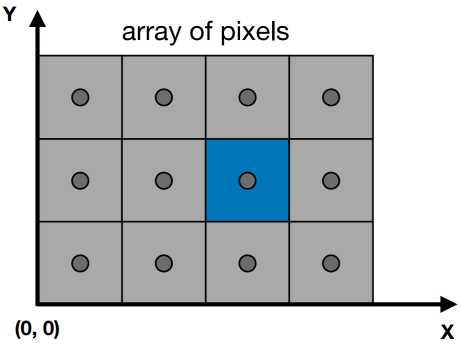
Rasterization

The verb *raster* comes from German *Raster*, literally "screen". *Rasterization*, the process, refers to the task of converting continuous shapes described in real vectors to discrete raster images, most notably with square pixels for output display. So far, we have successfully transformed all vector primitives to the Normalized Device Coordinates (NDC) with a series of transformations. In fact, this is already the bulk of the work done. We discuss next the remaining few steps to raster a humble triangle, which in abundant quantity builds up models of unlimited details.

Viewport Transformation

A viewport transformation $\mathcal{U}$ involves stretching and translation that maps all x-y slices of the canonical cube K to the screen space. Given a screen of height H and width W in units of pixels, we define the 2D screen-space coordinate following the convention that

- The origin of the screen space is at the bottom-left of the display.
- The y axis of the screen-space coordinate points upward and the x component rightward to the display.
- Each pixel $(i, j),\ i \in [0..W), j \in [0..H)$ is a unit square centered at $(x_i, x_j) = (i + 0.5, j + 0.5)$ in screen space.



Under this setup, we can represent $\mathcal{U}$ easily in the matrix form as

$$\mathbf{U} = \begin{bmatrix} \frac{W}{2} & 0 & 0 & \frac{W}{2} \\ 0 & \frac{H}{2} & 0 & \frac{H}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Note that we leave the z component untouched since it will be helpful later (Z-buffer) in deciding occlusion when triangles overlap in screen space.

Rasterizing a triangle

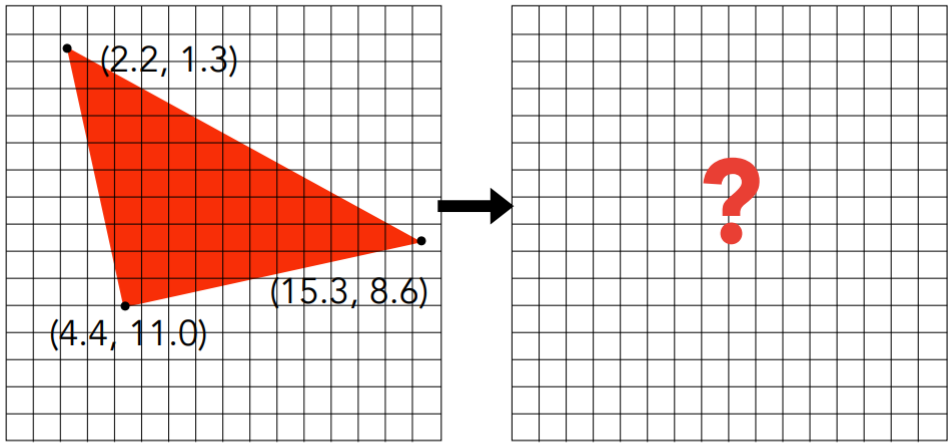
Why triangles?

Triangles have a number of nice properties as a fundamental geometric primitive:

- It breaks up other planer shapes (polygons).
- It is guaranteed to be planar (Three non-colinear points uniquely define a plane)
- It has a well-defined interior (Easy to test algorithmically, up next)
- It is easy to interpolate between vertices (Barycentric coordinates, up next)

Rasterization = discrete sampling

What Pixel Values Approximate a Triangle?



Input: position of triangle vertices projected on screen

Output: set of pixel values approximating triangle

Suppose we'd like to rasterize a red triangle T with vertices already transformed to screen space, as shown on the left. Surely, it is impossible to be fully authentic here due to the finitely many pixels at our disposal. The question is, then, how should we set the value of each pixel on the right so that the raster image on the right resemble the original triangle?

A straightforward idea is to *sample* at discrete locations, say at the center of each pixel, and color the pixel red if the sampled location is inside the triangle. Formally, this amounts to discrete evaluation of the indicator function

$$I_T(x,y) = \begin{cases} 1 & : \text{point } (x,y) \text{ inside } T \\ 0 & : \text{otherwise} \end{cases} \tag{2}$$

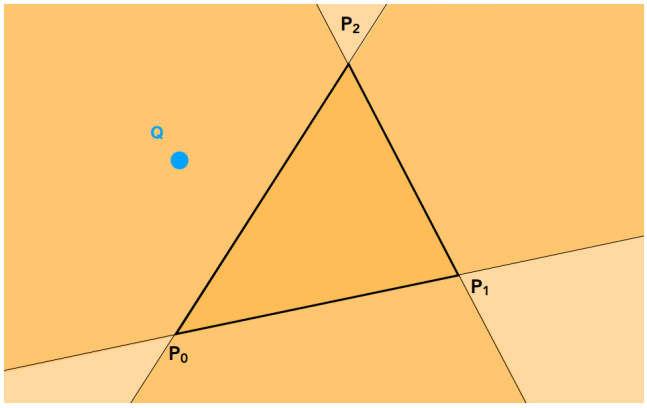
at locations $(i+0.5, j+0.5)$ for all pixels (i,j).

It begs the question: How do we actually evaluate I_T? As we shall see next, this can be done by a straightforward application of the cross product.

Deciding the interior of a triangle

Suppose the three vertices of the triangle T are $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ in screen-space coordinates. Imagine a truck traveling along the boundary of the triangle, say $\mathbf{p}_0 \to \mathbf{p}_1 \to \mathbf{p}_2 \to \mathbf{p}_0$, the critical observation is that any point inside the triangle necessarily stays on the *same side* of the truck (In our case, it stays on the left), whereas any point outside the triangle must see the truck on the left at some moment, and on the right at another.

See the following figure for an example, where the left half-plane of each directed path $(\mathbf{p}_0\mathbf{p}_1, \mathbf{p}_1\mathbf{p}_2, \mathbf{p}_2\mathbf{p}_0)$ is shaded yellow. The interior of the triangle exactly matches the triply-shaded region.



How should we decide if a point $\mathbf{q}$ is on the *left* of the path $\mathbf{p}_i\mathbf{p}_j$? This happens when the angle between $\mathbf{p}_i\mathbf{p}_j$ and $\mathbf{p}_i\mathbf{q}$ is in the range $(0, \pi)$. In other words, $(\mathbf{p}_i\mathbf{p}_j \times \mathbf{p}_i\mathbf{q})_z > 0$. Enforcing this for all path of the triangle, we must have

$$(\mathbf{p}_i\mathbf{p}_j \times \mathbf{p}_i\mathbf{q})_z > 0 \quad \forall \text{path } \mathbf{p}_i\mathbf{p}_j \tag{3}$$

If the order of the vertices is arbitrary, we also need to account to the possibility that the truck travels in the reversed order $\mathbf{p}_0 \to \mathbf{p}_2 \to \mathbf{p}_1 \to \mathbf{p}_0$, so that an interior point $\mathbf{q}$ is always on the *right* of the each path. This amounts to the condition that

$$(\mathbf{p}_i\mathbf{p}_j \times \mathbf{p}_i\mathbf{q})_z < 0 \quad \forall \text{path } \mathbf{p}_i\mathbf{p}_j \tag{4}$$

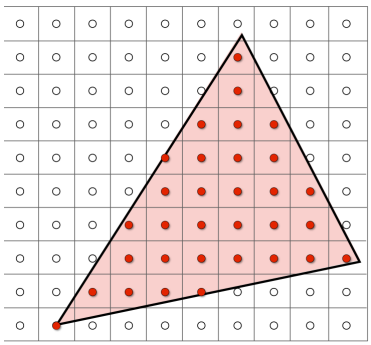
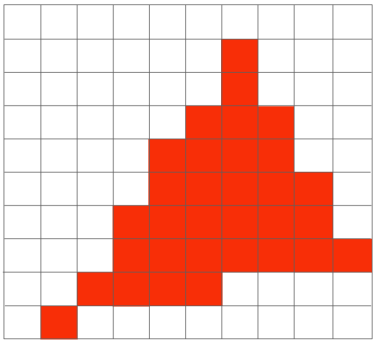
In summary, we have the convenient characterization

$$\text{point } (x, y) \text{ inside } T \iff (\mathbf{p}_i\mathbf{p}_{(i+1)\text{mod}3} \times \mathbf{p}_i\mathbf{q})_z \text{ have same sign for } i = 0, 1, 2 \tag{5}$$

where $\mathbf{q} = (x, y, 0)$.

Antialiasing

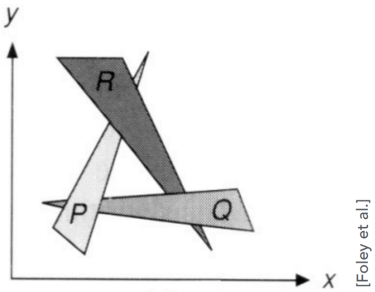
The basic idea of sampling looks simple. Nonetheless, a closer inspection of the rasterization result may bring unsettling news: The raster image of the triangle definitely looks a lot more jagged than ideal. What went wrong?

Pixel-center sampling a triangle	Raster image
	

We can use sampling theory to explain this phenomenon. As discussed earlier, a triangle in screen space can be thought of as an indicator function I on the domain $\mathbb{R}^2$. Due to its finite support, I is a band-limited signal. [WIP]

Z-buffering

Different algorithms exist for handling occlusion among triangles. The *painter's algorithm* assigns a single depth to each triangle, sorts them, and draw the triangles back to front. Since each triangle owns a single depth, painter's algorithm cannot handle the scenario in which a portion of the triangle A is behind triangle B and the rest in front.



[Foley et al.]

Fig: Overlapping triangles. Impossible to draw individually based on single depth

The z-buffering algorithm solves the problem by maintaining two buffers, one called the *frame buffer* storing color values, and another called the *depth buffer* (or *z-buffer*) that stores depth values. The algorithm draws each triangle by iterating over the sample within the triangle, comparing depth of individual sample with the recorded smallest, and optionally updating both buffers if the new depth is smaller (i.e., the new sample is in front of the old one).

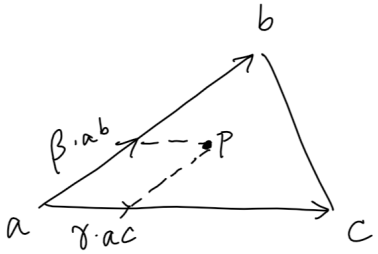
```
/* Z-buffering Algorithm */
foreach sample index (x, y):
    depthBuf[x, y] = ∞
    frameBuf[x, y] = BG_COLOR

foreach triangle T:
    foreach sample (x, y) in T:
        z = T.getDepth(x, y)     /* Interpolate depth */
        if z < depthBuf[y, x]:   /* new sample is in front */
            rgb = T.getColor(x, y, z)  /* Interpolate color */
            frameBuf[y, x] = rgb /* Update color */
            depthBuf[y, x] = z   /* Update depth */
```

Assuming each triangle T covers a constant number of samples, it is not hard to see the z-buffering algorithm takes $O(n)$ time for n triangles.

Barycentric coordinates (triangular interpolation)

We've previously set aside the topic of interpolating values within a triangle. For instance, what should happen in `T.getColor(x, y, z)` in the code above if we set three vertices of a triangle to distinct colors instead of a uniform red? A natural response would be that it should be a linear blend of the three vertex colors. Formally, given a triangle T with vertices $\mathbf{a}, \mathbf{b}, \mathbf{c}$, we should be able to express any point $\mathbf{p}$ in the triangle as a linear combination of $\mathbf{a}, \mathbf{b}, \mathbf{c}$.



Indeed, if we set $\mathbf{ab}$ and $\mathbf{ac}$ as two basis vectors centered at $\mathbf{a}$, then any point $\mathbf{p}$ inside triangle T can be expressed as

$$
\begin{aligned}
\mathbf{p} &= \mathbf{a} + \beta \cdot \mathbf{ab} + \gamma \cdot \mathbf{ac} \qquad \text{where } 0 \le \beta + \gamma \le 1 \\
&= \mathbf{a} + \beta \cdot (\mathbf{b} - \mathbf{a}) + \gamma \cdot (\mathbf{c} - \mathbf{a}) \\
&= (1 - \beta - \gamma)\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}
\end{aligned}
$$

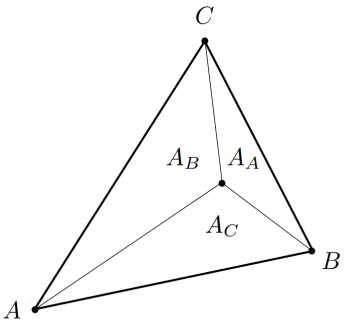
Writing $\alpha = 1 - \beta - \gamma$, we have

$$
\begin{aligned}
\boxed{\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}} \\
\text{where } \alpha, \beta, \gamma \ge 0 \\
\alpha + \beta + \gamma = 1
\end{aligned}
$$

One may identify this as a *convex combination* of $\mathbf{a}, \mathbf{b}, \mathbf{c}$. In the context of triangles, *barycentric coordinates* is a more popular name.

Cartesian-to-barycentric conversion

In practice, interpolating at a point within a triangle requires conversion from the Cartesian coordinates to the barycentric system. We utilize the relation of the barycentric coordinates with the proportion of triangular areas, shown below:



$$
\begin{aligned}
\alpha &= \frac{A_A}{A_A + A_B + A_C} \\
\beta &= \frac{A_B}{A_A + A_B + A_C} \\
\gamma &= \frac{A_C}{A_A + A_B + A_C}
\end{aligned}
$$

Suppose $\mathbf{P} \in \mathbb{R}^3$ is the point of interpolation with Cartesian coordinates (x, y, z). And the vertices of the triangle are $\mathbf{A} = (a_x, a_y, a_z), \mathbf{B} = (b_x, b_y, b_z), \mathbf{C} = (c_x, c_y, c_z)$. Observe that if we drop the z coordinate of the point and that of the vertices, the barycentric coordinates of the point $\mathbf{p} = (x, y)$ w.r.t. the vertices $\mathbf{a} = (a_x, a_y), \mathbf{b} = (b_x, b_y), \mathbf{c} = (c_x, c_y)$ are the same as $\mathbf{P}$'s since all areas change by a constant factor. Hence

$$
\begin{aligned}
\alpha &= \frac{\det(\mathbf{bc}|\mathbf{bp})}{S} = \frac{(y - b_y)(c_x - b_x) - (x - b_x)(c_y - b_y)}{S} \\
\beta &= \frac{\det(\mathbf{ca}|\mathbf{cp})}{S} = \frac{(y - c_y)(a_x - c_x) - (x - c_x)(a_y - c_y)}{S} \\
\gamma &= 1 - \alpha - \beta
\end{aligned}
$$

where $S = \det(\mathbf{ab}|\mathbf{ac}) = (b_x - a_x)(c_y - a_y) - (c_x - a_x)(b_y - a_y)$ is the area of the triangle.

References

[1] https://sites.cs.ucsb.edu/~lingqi/teaching/games101.html

Next: Shading I

To be continued.