WIKIBOOKS

GLSL Programming/Rasterization

Rasterization is the stage of the OpenGL (ES) 2.0 pipeline that determines the pixels covered by a primitive (e.g. a triangle) and interpolates the output variables of the vertex shader (i.e. varying variables and depth) for each covered pixel. These interpolated varying variables and the interpolated depth are then given to the fragment shader. (In a wider sense, the term "rasterization" also includes the execution of the fragment shader and the per-fragment operations.)

Contents

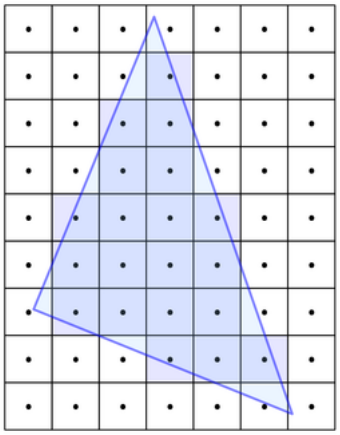
Usually, it is not necessary for GLSL programmers to know more about the rasterization stage than described in the previous paragraph. However, it is useful to know some details in order to understand features such as perspectively correct interpolation and the role of the fourth component of the vertex position that is computed by the vertex shader. For some advanced algorithms in computer graphics it is also necessary to know some details of the rasterization process.

The main two parts of the rasterization are:

- Determining the pixels that are covered by a primitive (e.g. a triangle)
- Linear interpolation of varying variables and depth for all covered pixels

Determining Covered Pixels

In OpenGL (ES), a pixel of the framebuffer is defined as being covered by a primitive if the center of the pixel is covered by the primitive as illustrated in the diagram to the right.

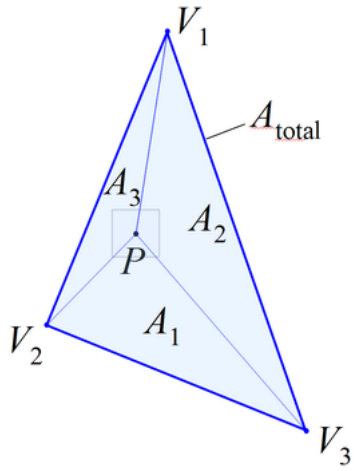


Pixels covered by a triangle.

There are certain rules for cases when the center of a pixel is exactly on the boundary of a primitive. These rules make sure that two adjacent triangles (i.e. triangles that share an edge) never share any pixels (unless they actually overlap) and never miss any pixels along the edge; i.e. each pixel along the edge between two adjacent triangles is covered by either triangle but not by both. This is important to avoid holes and (in case of semitransparent triangles) multiple rasterizations of the same pixel. The rules are, however, specific to implementations of GPUs; thus, they won't be discussed here.

Linear Interpolation of Varying Variables

Once all the covered pixels are determined, varying variables are interpolated for each pixel. For simplicity, we will only discuss the case of a triangle. (A line works like a triangle with two vertices at the same position.)



Areas in a triangle that are used to interpolate varying variables.

For each triangle, the vertex shader computes the positions of the three vertices. In the diagram to the right, these positions are labeled as V_1, V_2, and V_3. The vertex shader also computes values of varying variables at each vertex. We denote one of them as f_1, f_2, and f_3. Note that these values refer to the same varying variable computed at different vertices. The position of the center of the pixel for which we want to interpolate the varying variables is labeled by P in the diagram.

We want to compute a new interpolated value f_P at the pixel center P from the values f_1, f_2, and f_3 at the three vertices. There are several methods to do this. One is using barycentric coordinates α_1, α_2, and α_3, which are computed this way:

$$\begin{aligned}
\alpha_1 &= \frac{A_1}{A_{\text{total}}} = \frac{\text{area of triangle } PV_2V_3}{\text{area of triangle } V_1V_2V_3} \\
\alpha_2 &= \frac{A_2}{A_{\text{total}}} = \frac{\text{area of triangle } V_1PV_3}{\text{area of triangle } V_1V_2V_3} \\
\alpha_3 &= \frac{A_3}{A_{\text{total}}} = \frac{\text{area of triangle } V_1V_2P}{\text{area of triangle } V_1V_2V_3}
\end{aligned}$$

The triangle areas A_1, A_2, A_3, and A_{total} are also shown in the diagram. In three dimensions (or two dimensions with an additional third dimension) the area of a triangle between three points Q, R, S, can be computed as one half of the length of a cross product:

$$\text{area of triangle } QRS = \tfrac{1}{2}|\overrightarrow{QR} \times \overrightarrow{QS}|$$

With the barycentric coordinates α_1, α_2, and α_3, the interpolation of f_P at P from the values f_1, f_2, and f_3 at the three vertices is easy:

$$f_P = \alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3$$

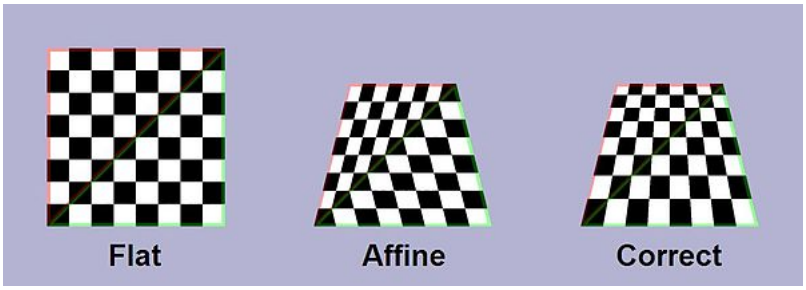
This way, all varying variables can be linearly interpolated for all covered pixels.

Perspectively Correct Interpolation of Varying Variables

The interpolation described in the previous section can result in certain distortions if applied to scenes that use perspective projection. For the perspectively correct interpolation the distance to the view point is placed in the fourth component of the three vertex positions (w_1, w_2, and w_3) and the following equation is used for the interpolation:

$$f_P = \frac{\alpha_1 f_1/w_1 + \alpha_2 f_2/w_2 + \alpha_3 f_3/w_3}{\alpha_1/w_1 + \alpha_2/w_2 + \alpha_3/w_3}$$

Thus, the fourth component of the position of the vertices is important for perspectively correct interpolation of varying variables. Therefore, it is also important that the perspective division (which sets this fourth component to 1) is not performed in the vertex shader, otherwise the interpolation will be incorrect in the case of perspective projections. (Moreover, clipping fails in some cases.)



Comparison of perspectively incorrect interpolation of texture coordinates (labeled “Affine”) and perspectively correct interpolation (labeled “Correct”).

It should be noted that actual OpenGL implementations are unlikely to implement exactly the same procedure because there are more efficient techniques. However, all perspectively correct linear interpolation methods result in the same interpolated values.

Further Reading

All details about the rasterization of OpenGL ES are defined in full detail in Chapter 3 of the “OpenGL ES 2.0.x Specification” available at the “Khronos OpenGL ES API Registry” (https://www.khronos.org/registry/gles/).

< GLSL Programming