Shading with GLSL

Vertex, Fragment Shaders & the Shading Equation

Francesco Andreussi

Bauhaus–Universität Weimar

10 December 2019

Bauhaus-Universität Weimar

Faculty of Media

Shaders Recap

The Shaders are additional programs necessary to the rendering pipeline.

Some shaders, such as the Geometry and the Tessellation, are optional while the Vertex and the Fragment are **BOTH compulsory**.

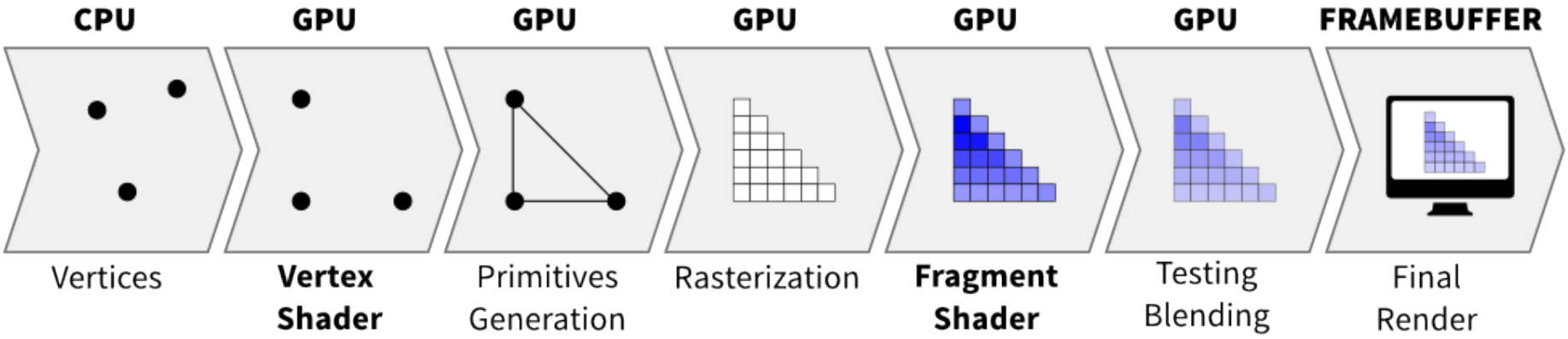


Fig. 1: The Rendering Pipeline (again)

Vertex Shader

The **Vertex Shader** processes one vertex at a time and its main function is to compute the vertex position in **Clip Space**.

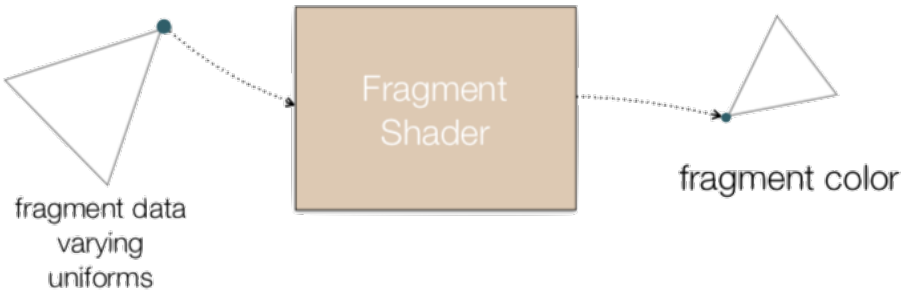
Fig. 2: An essential Vertex Shader

Therefore, a Vertex Shader must have as input the vertex position (in object space). This is a **Varying** (it varies from vertex to vertex) as well as all the other vertex–specific attributes (e.g. the colour).

All the others inputs are called **Uniforms** and they are constant for all the vertices processed by a specific shader.

The varying **Outputs** of a Vertex Shader are forwarded throughout the pipeline.

Fragment Shader



The **Fragment Shader** process one fragment at a time and outputs (at least) the color of that fragment.

Fig. 3: An essential Fragment Shader

Its minimum input is the value of the fragment position in viewport space (generated in the rasterisation step), which is a **Varying** (it varies for each fragment), as well as all the other data generated by *interpolating* the output values of the Vertex Shader, which are also generated during the rasterisation.

Here, as in every other shader, it is possible to have **Uniforms**: values constant for all the fragments processed by a specific shader.

The varying **Outputs** are passed to the next steps of the pipeline.

The Shading Equation

Theoretically, we'd like to compute the color (aka light) emitted and/or reflected in every direction from every "piece" of material:

$$L_o(X, \overrightarrow{v}) = \int_{\|S^2\|} L_i(Y, \overrightarrow{l}) |\overrightarrow{l} \cdot \overrightarrow{n}| f(\overrightarrow{v}, \overrightarrow{l}) \, \mathrm{d}l \, .$$

That is practically not achievable, so the formula is made computable in this way:

$$L_o(X, \overrightarrow{v}) = A + \sum_{Y \in lights} \beta(Y, X) |\overrightarrow{l} \cdot \overrightarrow{n}| f(\overrightarrow{v}, \overrightarrow{l}) \, ,$$

where A is the *Ambient term*, β is the *incoming light intensity function* and f is the *BRDF*.

Ambient Term & Point Lights

The Ambient term is only a color (usually really dark) used in order to light up a bit the shadowed parts of an object that, otherwise, would be completely black since the indirect lights are not taken in account.

Since $L_i(Y, \overrightarrow{l})$ do not account for the decay of the light power, we use instead β that is the intensity of the light that reaches the processed fragment.

$$\beta(Y, X) = \frac{\phi}{4\pi\,|Y - X|^2},$$

where $\phi = \textit{lightColor} \cdot \textit{lightIntensity}$.
In this function the direction of the light does not matter because a *Point Light* emits the same amount of energy in every direction.

NOTE that in this approximation the Lights are not rendered; in fact, only the *effects* of the emitted light can be seen, and another Geometry has to be placed in order to signal the position of the Light.

Lambertian BRDF

A **Bidirectional Reflectance Distribution Function** is a function calculating how the light reflected from surface is distributed in the space. It determines the way of perceiving a surface affecting its appearance hence different materials have different BRDFs and/or the same function but with different parameters.

The Lambertian BRDF is the function for perfectly matte surfaces: the incoming light is reflected equally in every direction.

$$f(\overrightarrow{v}, \overrightarrow{l}) = \frac{\rho}{\pi},$$

where ρ is the "reflectivity" and $f(\overrightarrow{v}, \overrightarrow{l}) = 0, if\ |\overrightarrow{v} \cdot \overrightarrow{n}| < 0$ $or\ |\overrightarrow{l} \cdot \overrightarrow{n}| < 0$.

Blinn–Phong BRDF

Blinn–Phong (such as simple Phong) are, instead, useful to render plastic surfaces being able to decide its glossiness. The Blinn–Phong function is slightly more efficient, but also complex, than the Phong.

Phong: $f(\overrightarrow{v}, \overrightarrow{l}) = C_s(\overrightarrow{r} \cdot \overrightarrow{v})^{\alpha}$, where $\overrightarrow{r}$ is the reflected vector of $\overrightarrow{l}$ w.r.t. $\overrightarrow{n}$ and C_s is the specular color of the material.

Blinn–Phong: $f(\overrightarrow{v}, \overrightarrow{l}) = C_s(\overrightarrow{h} \cdot \overrightarrow{v})^{4\alpha}$, where $\overrightarrow{h} = \frac{\overrightarrow{l} + \overrightarrow{v}}{\| \overrightarrow{l} + \overrightarrow{v} \|}$ (the normalized $\overrightarrow{l} + \overrightarrow{v}$).

Hence, the complete Blinn–Phong Shading function is:

$$L_o(X, \overrightarrow{v}) = A + \sum_{Y \in lights} \beta(Y, X)(C_d(\overrightarrow{l} \cdot \overrightarrow{n})\frac{\rho}{\pi} + C_s(\overrightarrow{h} \cdot \overrightarrow{v})^{4\alpha}),$$

Interpolation Qualifiers

In GLSL it is possible to specify the way of interpolating the values that are generated by the vertex and are going to be used to the fragment shader.

The qualifiers are:

- **smooth:** the default, gives *perspective–correct* interpolation of the data,
- **flat:** the values are not interpolated at all,
- **non–perspective:** the data are linearly interpolated in window space.

The use is the following: `<qualifier> <in/out> varyingName`.
The `out` qualifiers of the Vertex Shader and the `in` ones of the Fragment Shader **must** match.

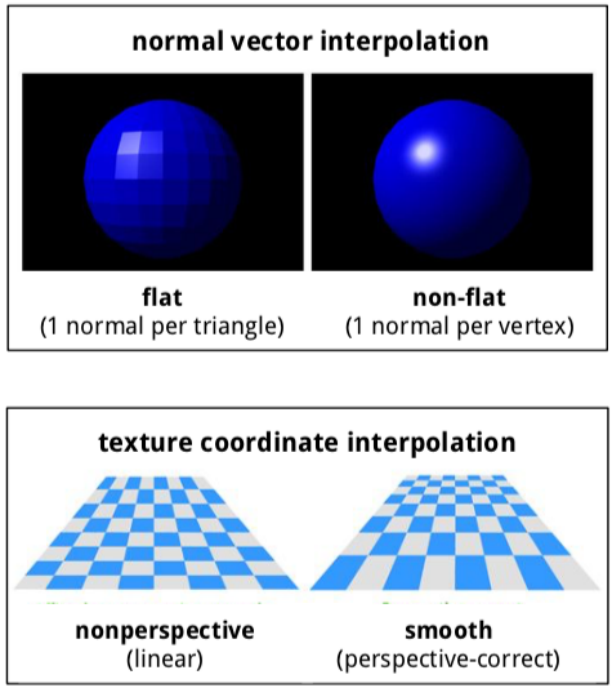
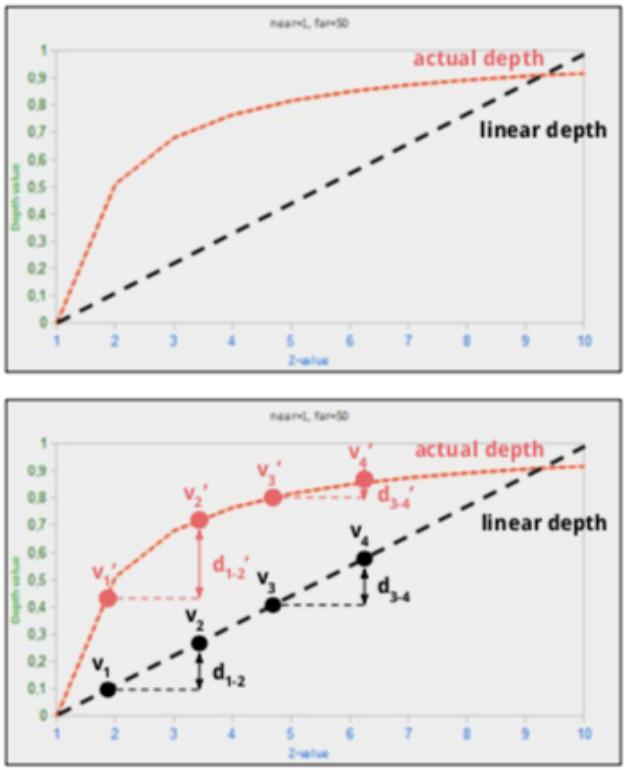


Fig. 4: Interpolation Examples

Perspective–Correct Interpolation

Since the z–coordinate is non–linear (in windows space) for the perspective, it is should be taken in account when interpolating the other values too.



- **linear:** $\dfrac{v_2 - v_1}{v_4 - v_3} = \dfrac{d_{1,2}}{d_{3,4}}$,
- **perspective–correct:** $\dfrac{v_2' - v_1'}{v_4' - v_3'} \neq \dfrac{d_{1,2}'}{d_{3,4}'}$.

For further information, check out this really short *paper*.

Fig. 5: Interpolation Differences

Thanks for the Attention!