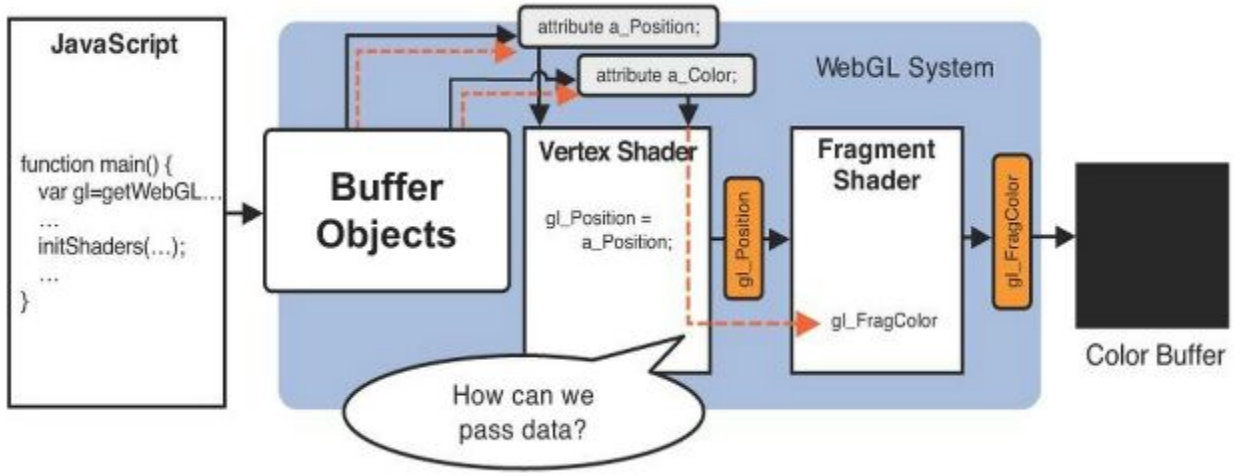
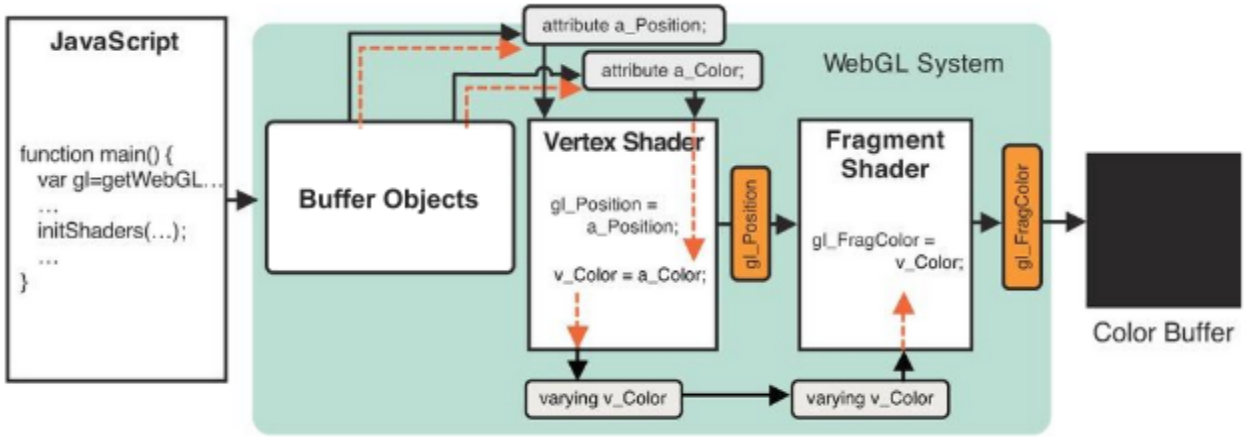
CMSC 161 UV-1L
Interactive Computer Graphics
Meeting 05 - Graphical Projection

Varying Variable

File: 06-varying01.html
06-varying01.html



Recall from previous examples that we are using uniform variable to set the color information to the vertex shader. However, that won't work on vertices that have their own separate color information.

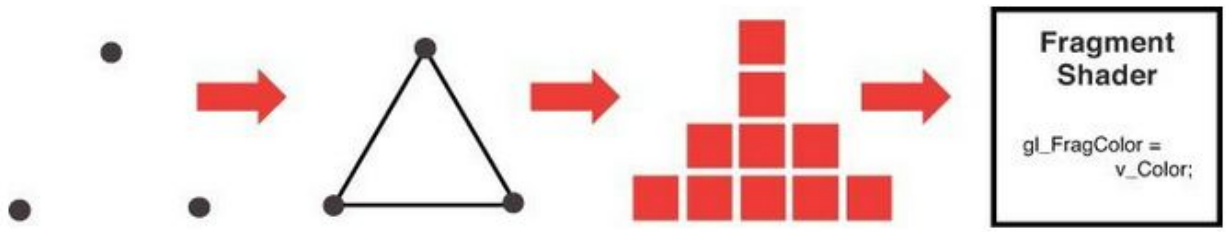


Varying variable

Varying is the third type of the storage qualifier for shader variables. Varying variables allows us to pass data from vertices to fragments. Varying variables that are connected to each other must be declared with the same name on both vertex and fragment shader.

```
varying <data_type> <variable name>;
```

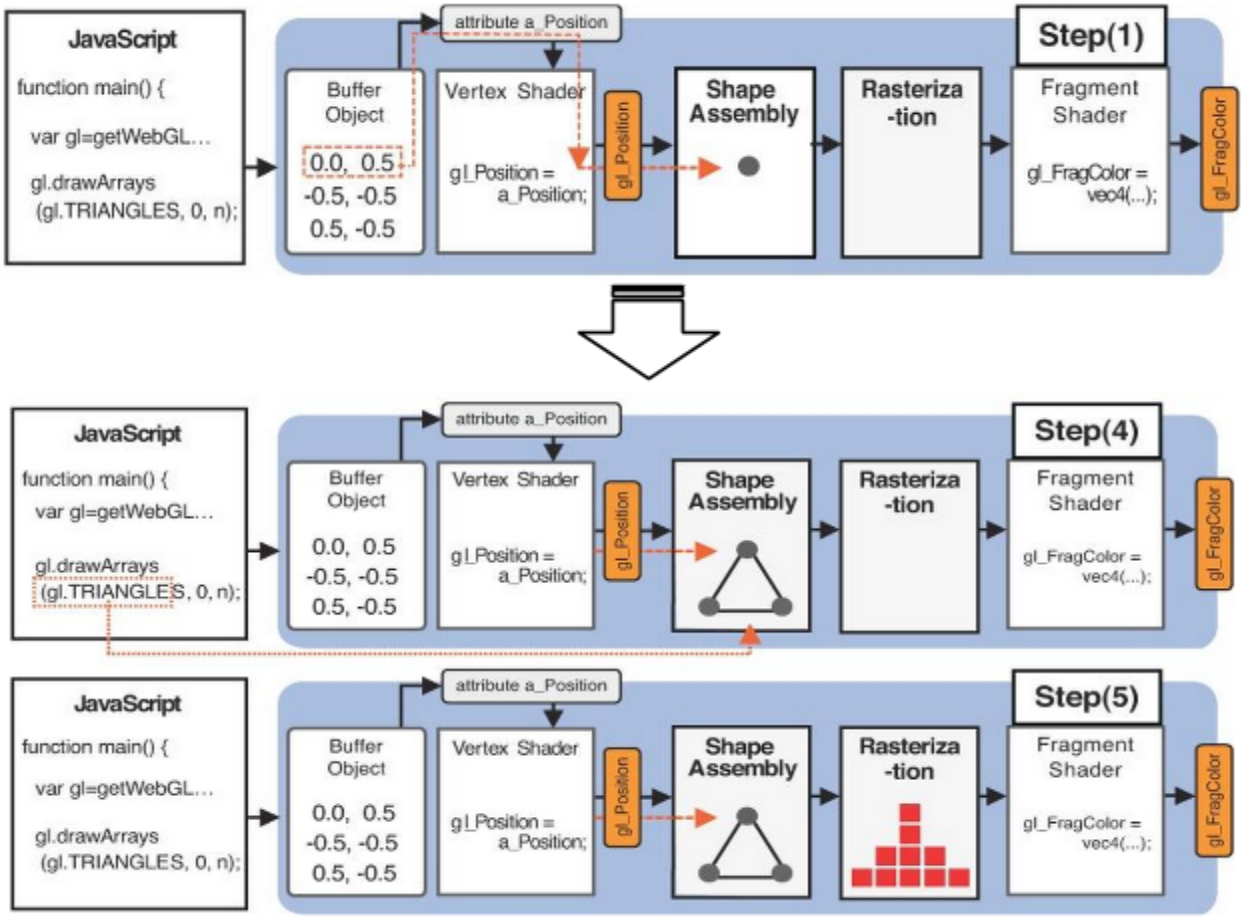
CMSC 161 UV-1L
Interactive Computer Graphics
Meeting 05 - Graphical Projection



There are two processes between vertex shader execution and fragment shader execution and that are:

Primitive Assembly (Shape Assembly)
The shape of the object from the specified vertices is assembled. The first argument of `gl.drawArrays()` is used to determine which type of primitive should be assembled.

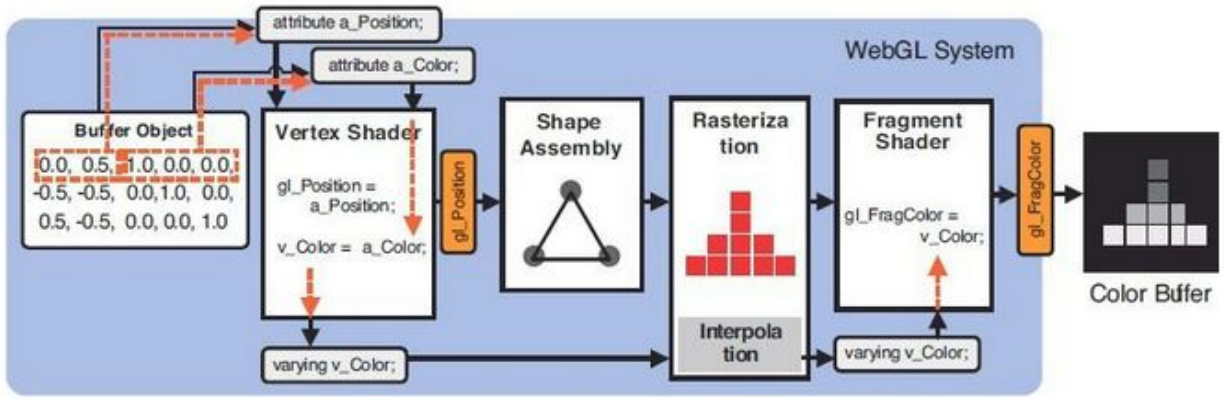
Rasterization
The primitive is converted into fragments. Each of the fragment will be run with its own fragment shader execution.

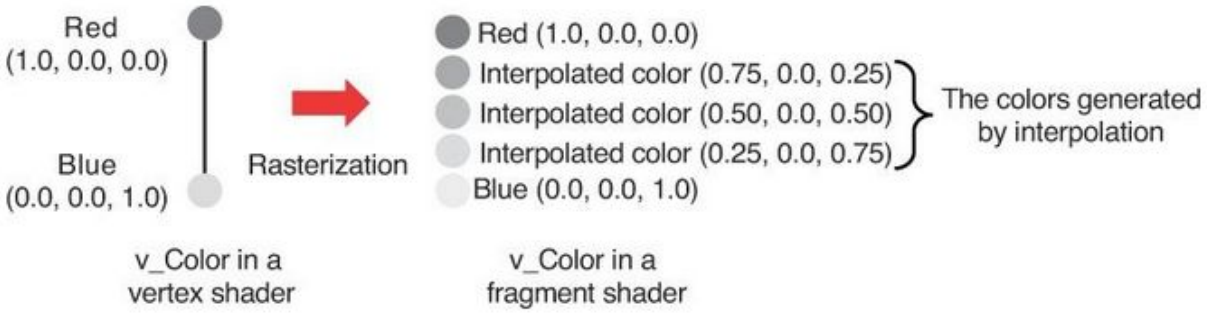


CMSC 161 UV-1L
Interactive Computer Graphics
Meeting 05 - Graphical Projection

Behavior of Varying Variables

The value assigned to the varying variable in the vertex shader is interpolated at the rasterization phase since there are more fragments than vertices. This is the reason why the storage qualifier is named varying.





CMSC 161 UV-1L
Interactive Computer Graphics
Meeting 05 - Graphical Projection

Alpha Blending (Compositing/Translucent)

File: 06-blending01.html
06-blending02.html
06-blending03.html

Alpha blending is the process of combining an image with a background to create the appearance of partial or full transparency. It is used to roughly emulate translucent objects in computer graphics.

Alpha blending is represented by the equation

$$FC = SF * SC + DF * DC$$

FINAL_COLOR = **source_factor** * source_color + **destination_factor** * destination_color

Specifically, the equation can be further dissected as follows:

$$FC_{red} = SF * SC_{red} + DF * DC_{red}$$
$$FC_{blue} = SF * SC_{blue} + DF * DC_{blue}$$
$$FC_{green} = SF * SC_{green} + DF * DC_{green}$$
$$FC_{alpha} = SF * SC_{alpha} + DF * DC_{alpha}$$

2 major steps to use alpha blending in webGL:

1. Enable the alpha blending function

```
gl.enable(gl.BLEND);
```

2. Specify the blending function

```
gl.blendFunc(source_factor,destination_factor);
gl.blendEquation(mode);
```

Constant factor parameters for gl.blendFunc()

Constant	Value
gl.ZERO	0.0
gl.ONE	1.0
gl.SRC_COLOR	SC

gl.ONE_MINUS_SRC_COLOR	$1 - SC$
gl.DST_COLOR	DC
gl.ONE_MINUS_DST_COLOR	$1 - DC$
gl.SRC_ALPHA	SC_{alpha}
gl.ONE_MINUS_SRC_ALPHA	$1 - SC_{alpha}$
gl.DST_ALPHA	DC_{alpha}
gl.ONE_MINUS_DST_ALPHA	$1 - DC_{alpha}$

Sample Blending Equation

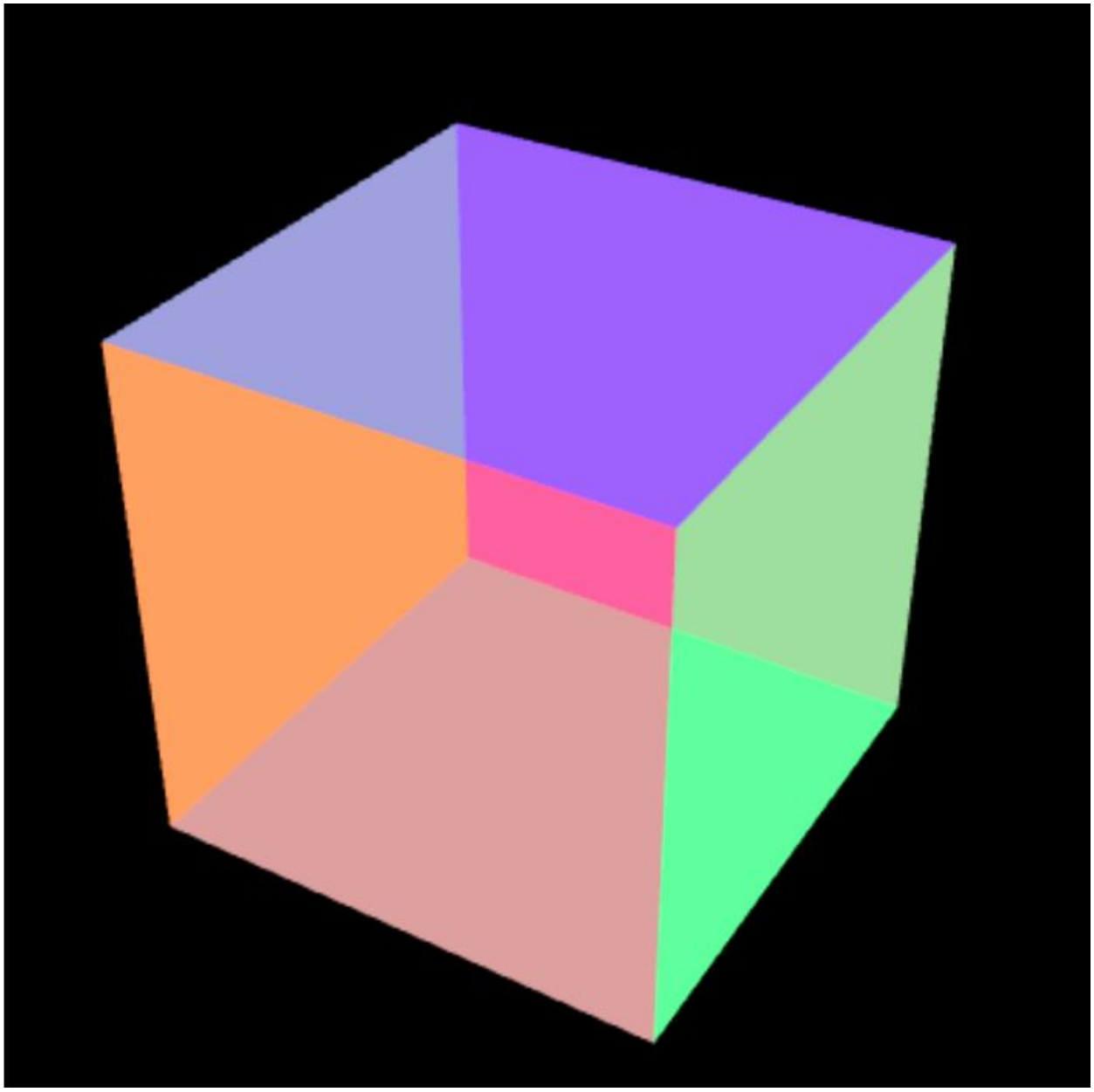
```
gl.blendFunc(gl.SRC_ALPHA, gl.ONE_MINUS_SRC_ALPHA);
gl.blendEquation(gl.FUNC_SUBTRACT);
```

$$FC_{red} = SC_{alpha} * SC_{red} + (1 - SC_{alpha}) * DC_{red}$$
$$FC_{blue} = SC_{alpha} * SC_{blue} + (1 - SC_{alpha}) * DC_{blue}$$
$$FC_{green} = SC_{alpha} * SC_{green} + (1 - SC_{alpha}) * DC_{green}$$
$$FC_{alpha} = SC_{alpha} * SC_{alpha} + (1 - SC_{alpha}) * DC_{alpha}$$

CMSC 161 UV-1L
Interactive Computer Graphics
Meeting 05 - Graphical Projection

Exercise

Using WebGL, draw a scene with a blended cube the looks like the image below



Scoring:

10 points - Non-rotating cube with its correct blending effects

+5 points - Rotating cube with correct blending intact

CMSC 161 UV-1L
Interactive Computer Graphics
Meeting 05 - Graphical Projection

Sample Rotated Blended Cube: