Varying Variables

Chapter 13

Affine Functions

Review
Appendix B

- Constant function $f(x) = k$
- Linear function $y = mx$
- Affine function $y = mx + n$

Affine Transformation

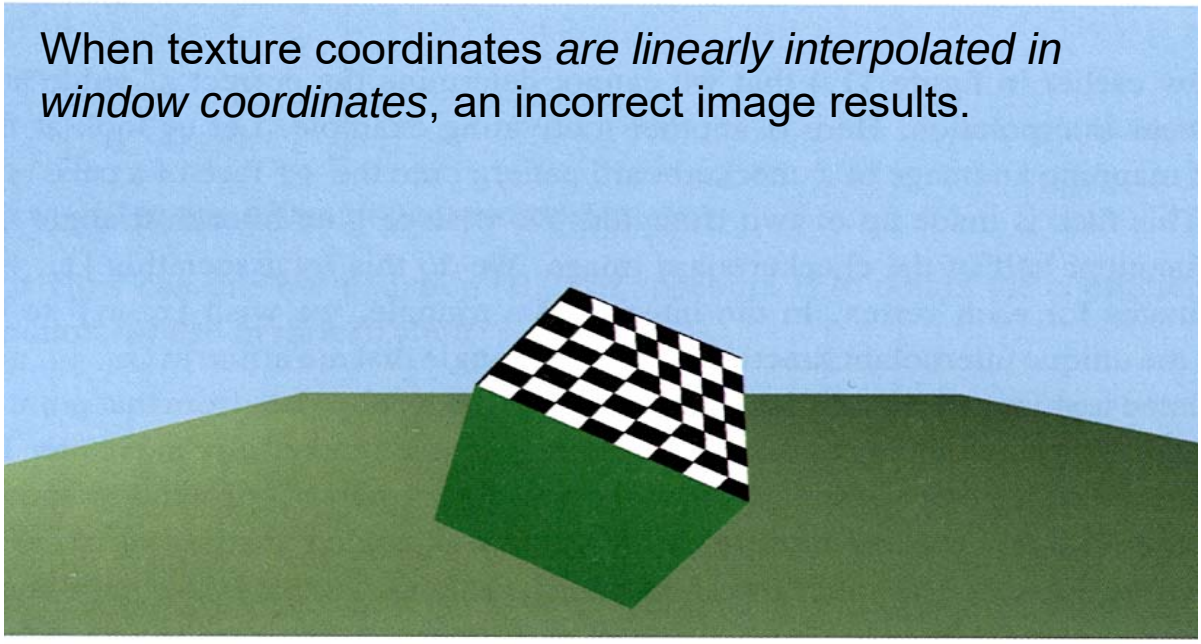
- An affine transformation is any transformation that preserves **collinearity** (i.e., all points lying on a line initially still lie on a line after transformation) and **ratios of distances** (e.g., the midpoint of a line segment remains the midpoint after transformation).
- Such a transformation is given by the formulas $x' = ax + by + p$ and $y' = cx + dy + q$ with the additional requirement that $ad - bc \neq 0$
- A linear transformation followed by a translation.
  Given a matrix M and a vector v,
  $$A(x) = Mx + v$$

Interpolation of varying variables

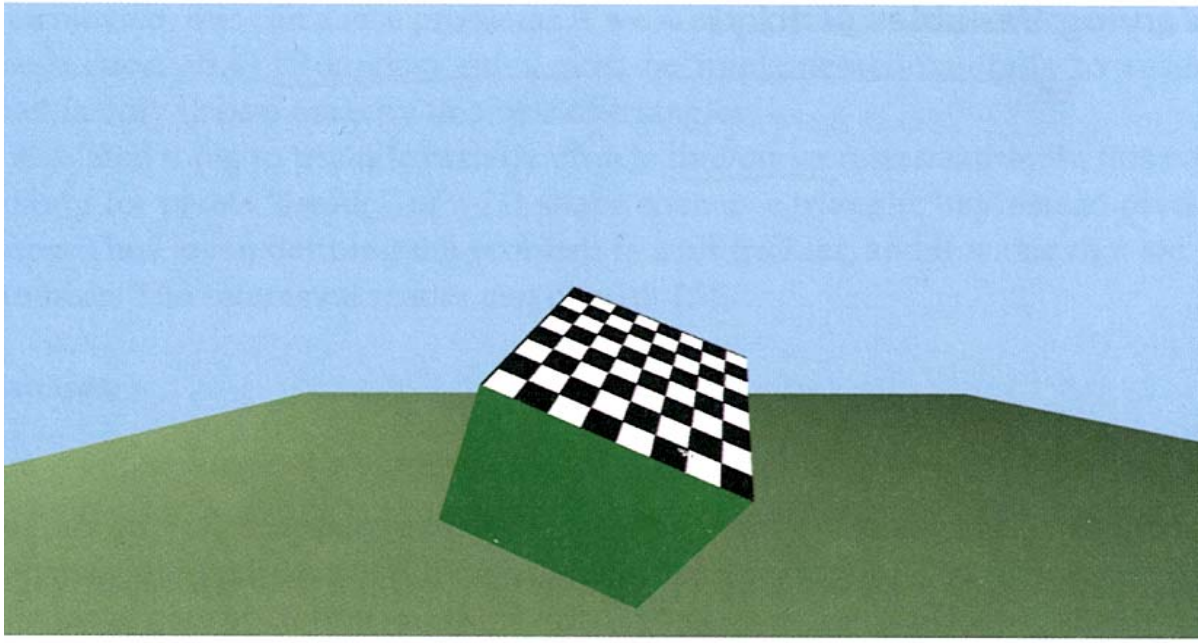
- In between the vertex and fragment shader, we need to interpolate the values of the varying variables.
- What is the desired interpolant, and how should we compute it.
- This is surprisingly subtle.

Wrong representation of texture

When texture coordinates *are linearly interpolated in window coordinates*, an incorrect image results.



Correct representation of texture



Motivation: texture coordinates

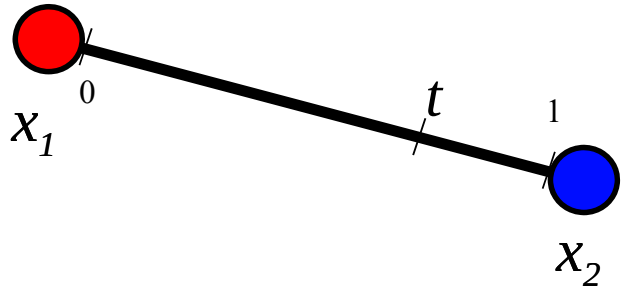
- Let us map a square checkerboard image onto a quad
- We break up the quad and the image each into two triangles.
- We will associate $[x_t, y_t]^t$ texture coordinates for each vertex.
- We desire that in the interior of a triangle, $[x_t, y_t]^t$ should be determined as the unique interpolant functions over the triangle that are affine in (x_o, y_o, z_o)

- If we use this interpolation and fetch the texture values we should get an expected foreshortening effect.

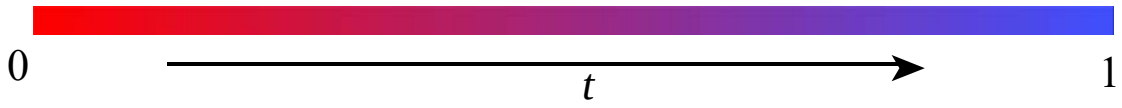
How the values defined at vertices are interpolated across the entire triangle?

Barycentric Interpolation

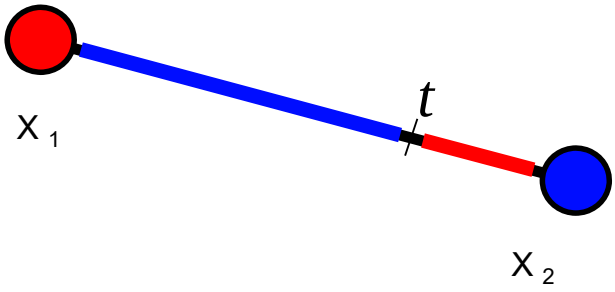
- How do you interpolate values defined at vertices across the entire triangle?



Want to define a value for every $t \in [0,1]$:

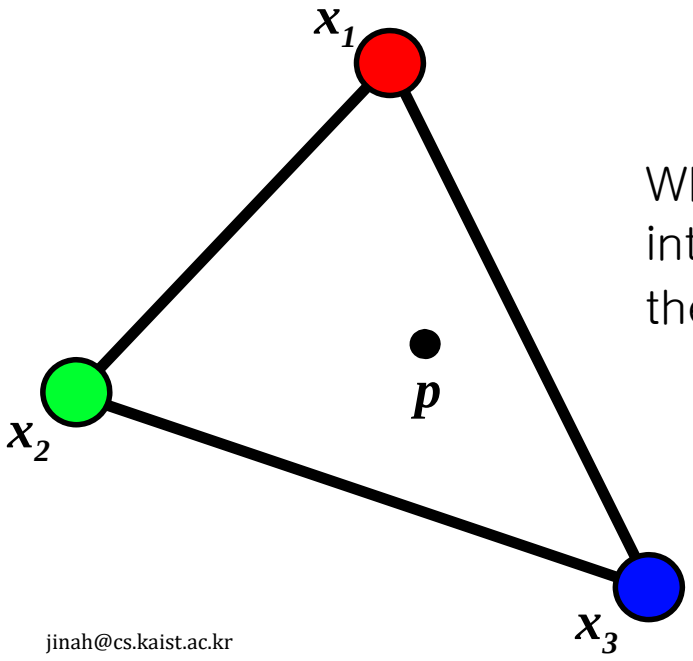


Percent blue = (length of blue segment)/(total length)
Percent red = (length of red segment)/(total length)

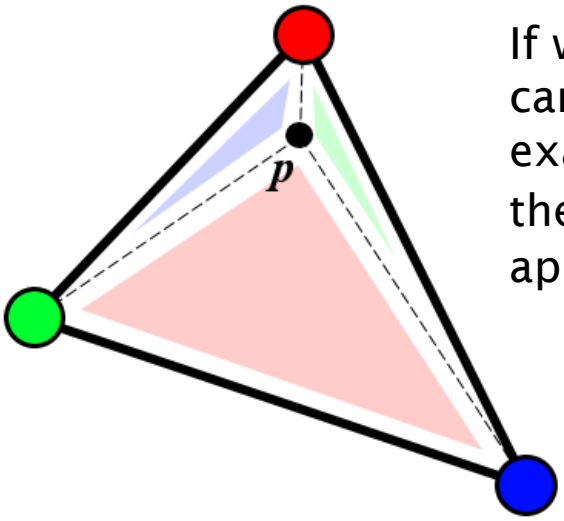


Percent blue = *t*
Percent red = *1-t*
Value at *t* = $t\,X_1 + (1-t)\,X_2$

Triangle



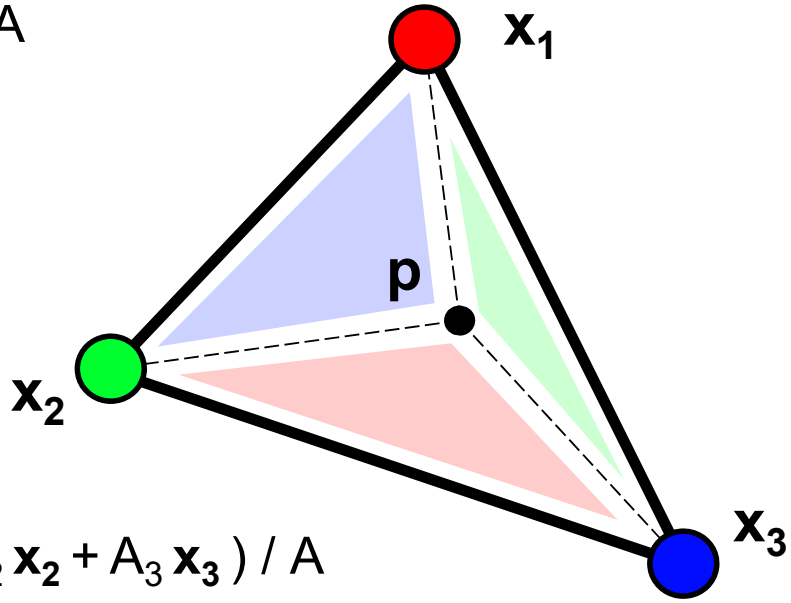
What's the interpolated value at the point $\boldsymbol{p}$?



If we color the areas carefully, the red area (for example) covers more of the triangle as $\boldsymbol{p}$ approaches the red point.

ratios of areas

- $A_1 + A_2 + A_3 = A$
- $r_1 = A_1/A$
  $r_2 = A_2/A$
  $r_3 = A_3/A$
- $r_1 + r_2 + r_3 = 1$



- $\mathbf{p} = (A_1 \mathbf{x_1} + A_2 \mathbf{x_2} + A_3 \mathbf{x_3}) / A$

Linear Interpolation

- Suppose we simply used linear interpolation on the x_t and y_t functions (on texture).
- Then, as we move by some fixed 2D vector displacement on the screen, the texture coordinates will be updated by some fixed 2D vector displacement in texture coordinates.
- In this case, all of the squares of the texture will map to equal size parallelograms.
- We will get an odd seam where the two triangles meet.

https://en.wikipedia.org/wiki/Texture_mapping

Perspective correctness

- Texture coordinates are specified at each vertex of a given triangle, and these coordinates are interpolated.
- If these texture coordinates are linearly interpolated across the screen, the result is affine texture mapping. This is a fast calculation, but there can be a noticeable discontinuity between adjacent triangles when these triangles are at an angle to the plane of the screen

Because affine texture mapping does not take into account the depth information about a polygon's vertices, where the polygon is not perpendicular to the viewer it produces a noticeable defect.

- **Perspective correct** texturing accounts for the vertices' positions in 3D space, rather than simply interpolating a 2D triangle.
- This achieves the correct visual effect, but it is slower to calculate.
- Instead of interpolating the texture coordinates directly, the coordinates are divided by their depth (relative to the viewer), and the reciprocal of the depth value is also interpolated and used to recover the perspective-correct coordinate.
- This correction makes it so that in parts of the polygon that are closer to the viewer the difference from pixel to pixel between texture coordinates is smaller (stretching the texture wider), and in parts that are farther away this difference is larger (compressing the texture).

- Affine texture mapping directly interpolates a texture coordinate u_a between two endpoints u_0 and u_1

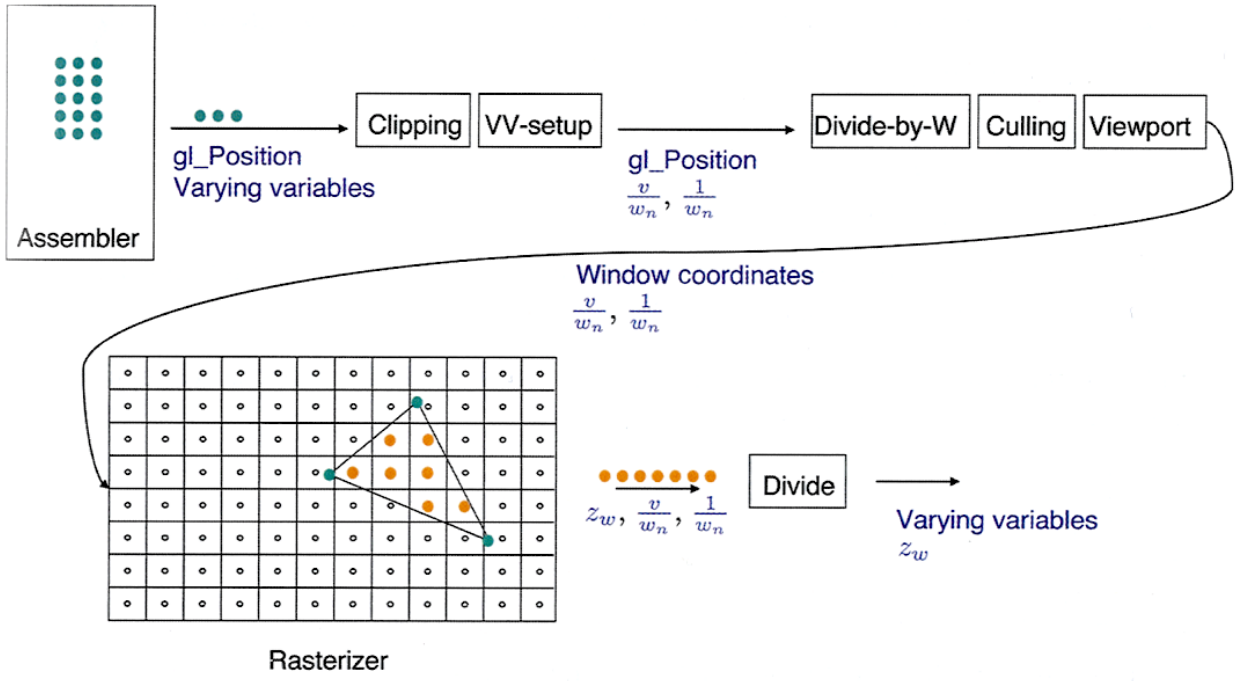
$$u_\alpha = (1-\alpha)u_0 + \alpha u_1 \text{ where } 0 \leq \alpha \leq 1$$

- Perspective correct mapping interpolates after dividing by depth z, then uses its interpolated reciprocal to recover the correct coordinate

$$u_\alpha = \frac{(1-\alpha)\frac{u_0}{z_0} + \alpha\frac{u_1}{z_1}}{(1-\alpha)\frac{1}{z_0}}$$

- All modern 3D graphics hardware implements perspective correct texturing

Interpolating varying variables

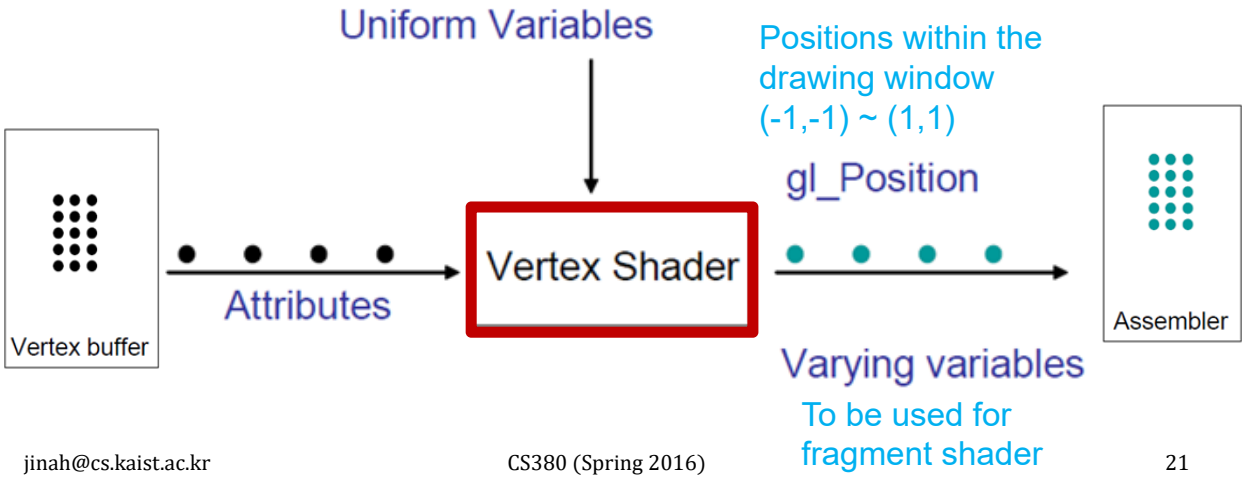


- For each vertex, and for each varying variable *v*, OpenGL creates an internal variable $\frac{v}{w_n}$.
- Additionally, for each vertex OpenGL creates one internal variable $\frac{1}{w_n}$.
- For each vertex, division is done to obtain the normalized device coordinates.
- For each vertex, the normalized device coordinates are transformed to window coordinates.

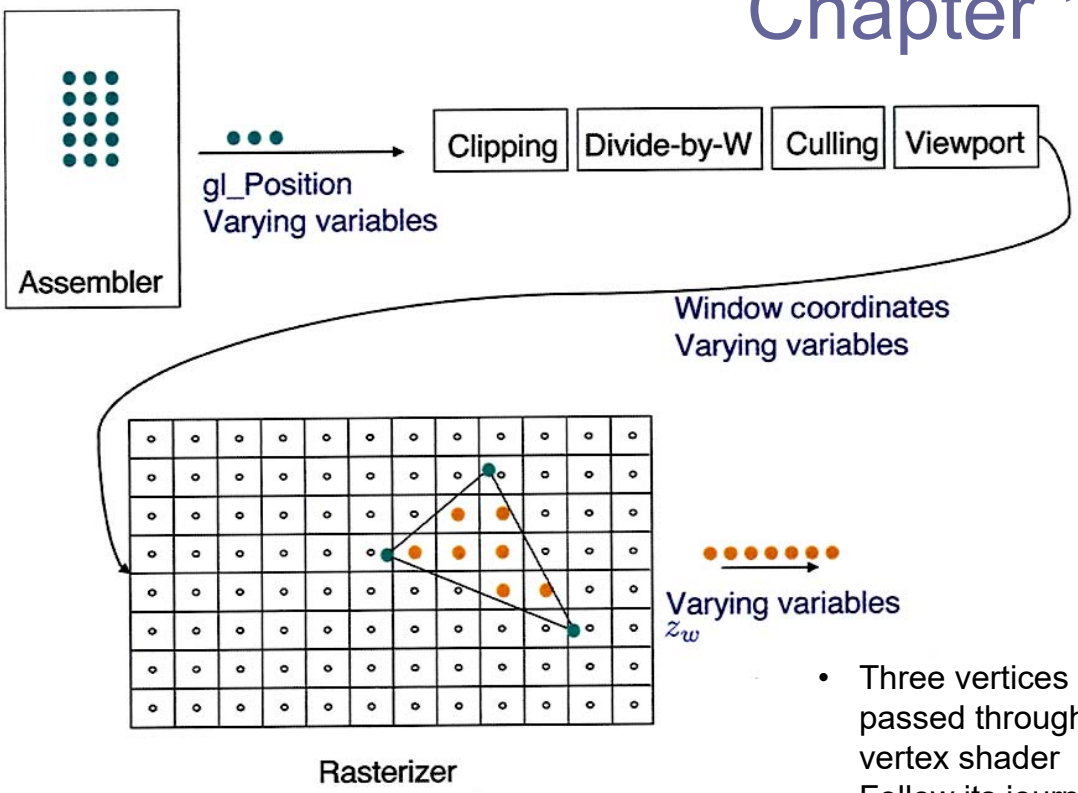
$$x_n = \frac{x_c}{w_c}, y_n = \frac{y_c}{w_c}, z_n = \frac{z_c}{w_c},$$

Chapter 2~6, 10

- Transformation
- Virtual camera
  - Mapping 3D coordinates to the actual 2D screen

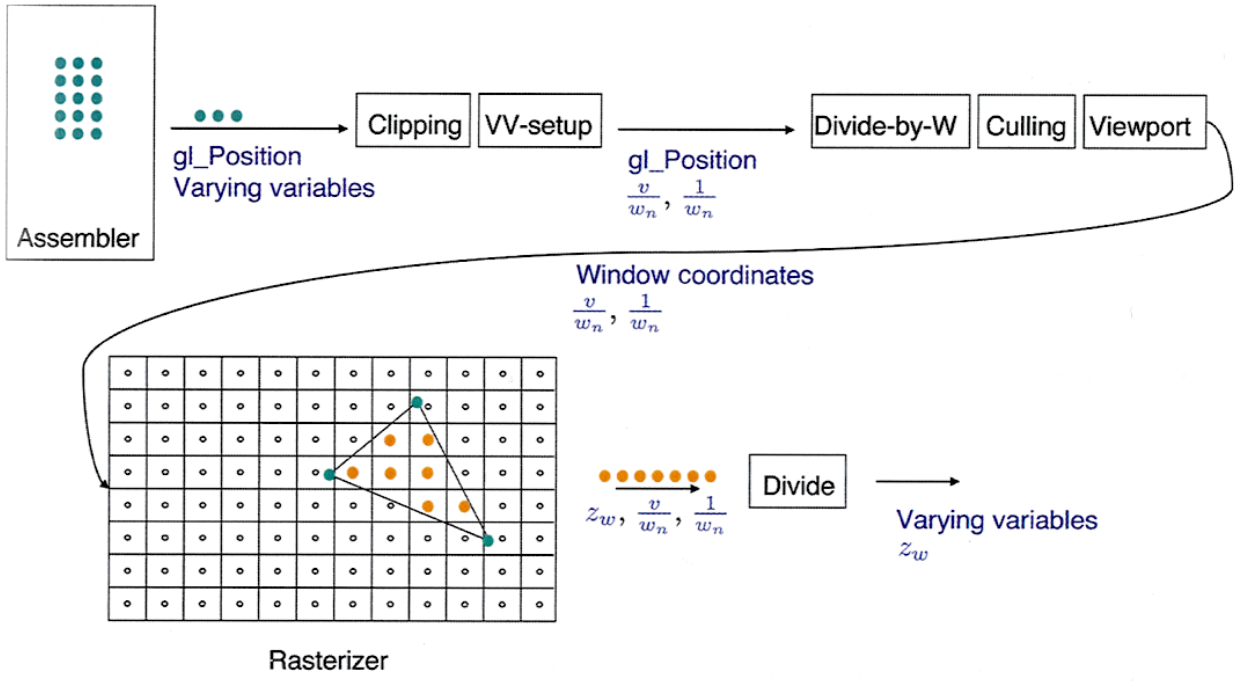


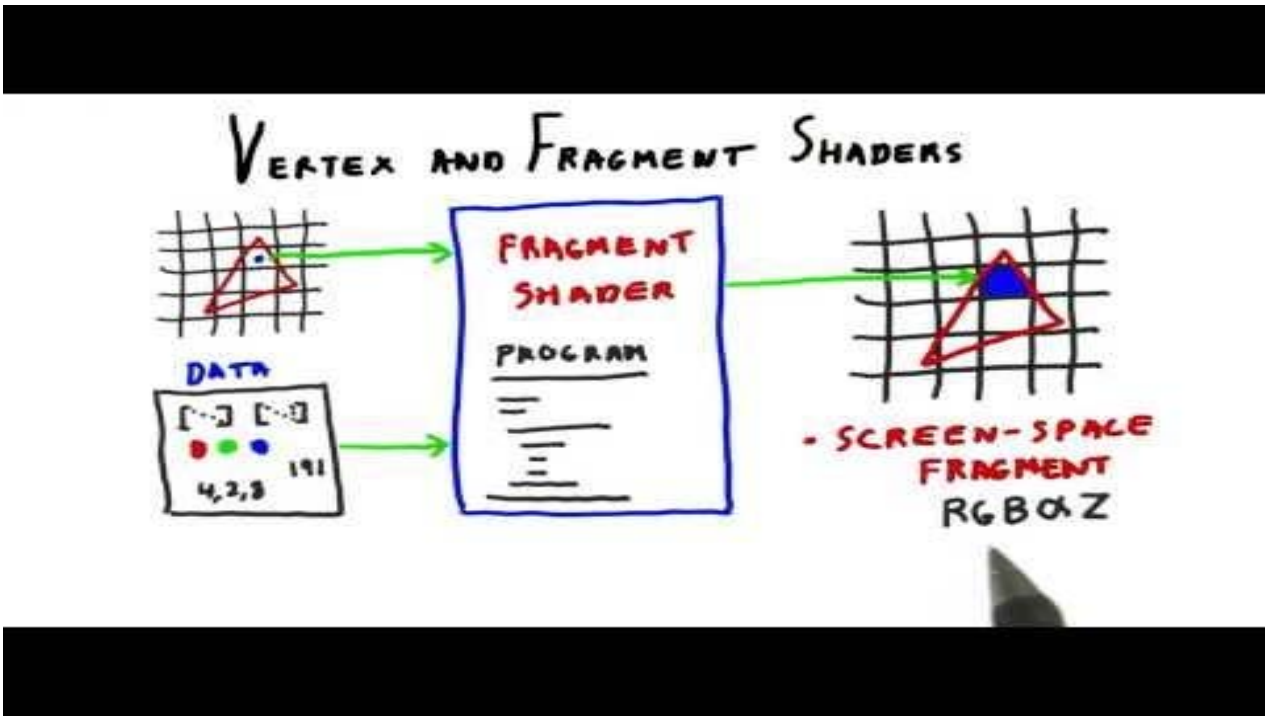
Chapter 11



- Three vertices have passed through the vertex shader
- Follow its journey to be a bunch of pixels

Chapter 12


Assembler
gl_Position
Varying variables
Clipping
VV-setup
gl_Position
$\frac{v}{w_n}, \frac{1}{w_n}$
Divide-by-W
Culling
Viewport
Window coordinates
$\frac{v}{w_n}, \frac{1}{w_n}$
Rasterizer
$z_w, \frac{v}{w_n}, \frac{1}{w_n}$
Divide
Varying variables
z_w


VERTEX AND FRAGMENT SHADERS
FRAGMENT SHADER PROGRAM
DATA
4,2,3
191
- SCREEN-SPACE FRAGMENT
RGBαZ