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.

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 \((x_o, y_o, z_o)\).
  - even steps on the geometry should be even steps in the texture
- if we use this interpolation and fetch the texture values we should get an expected foreshortening effect.

lerp

- suppose we simply used linear interpolation on the \(x_t\) and \(y_t\) functions as we move in window coordinates.
- 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 sized parallelograms.
- we will get an odd seam where the two triangles meet (see demo).
- see Fig 11.2: linear movement on the screen is non linear movement along the geometry, so should be non-linear movement along the texture.
- we would have the same problem if we tried to directly use linear interpolation on the eye coordiantes of each point in a triangle.

some terminology: Affine Functions

- We say a function \(f\) is an affine function in the variables \(x\) and \(y\) if it is of the form
  \[
  f(x, y) = ax + by + c
  \]
  for some constants \(a, b\) and \(c\).
- This can also be written as
  \[
  f = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
  \]
- also called “linear” but note the additive constant term.
- an affine function can be evaluated by plugging in \(x, y\) or by incremental evaluation along a line in \(x, y\) space.
- we already saw for example that \(z_n\) is an affine function of \(x_n, y_n\), and so was a 2d “edge function”.

affine/linear interpolation

- If we are given \(f_i\), the values of \(f\) for three (non collinear) points in the \((x, y)\) plane, say the vertices of a triangle,
  this determines an affine function \(f\) over the entire plane.
- In this case, we say that \(f\) is the linear interpolant of the values at the three vertices.
- The process of evaluating the linear interpolant of three vertices is called linear interpolation.
given the $f_i$, one can use some simple matrix operations to solve for the $[a, b, c]$.

**3D Affine**

- We say a function $f$ is affine in variables $x$, $y$, and $z$ if it is of the form
  \[ f(x, y, z) = ax + by + cz + d \]  
  (2)

- Such a function can be uniquely determined by its values at the four vertices of a tetrahedron sitting in 3D.

**triangles in 3d**

- Given a triangle in 3D, suppose we specify the value of a function at its three vertices.
- There may be many functions that are affine in $(x, y, z)$ that agree with these three values.
- But all of these functions will agree when restricted to the plane of the triangle.
- As such, we can refer to this restricted function over the triangle as the linear interpolant of the vertex values.

**some examples**

- during texture mapping, we will want each of the texture coordinates, $x_t$, $y_t$, as the unique interpolating functions over the triangle that are affine in $(x_o, y_o, z_o)$.
- when we associate colors with vertices, we will want our desired color field to be the unique interpolating function over the triangle that is affine in the object coordinates, $(x_o, y_o, z_o)$.
- As a rather self referential example, we can even think of each of the three object coordinates of a point on some triangle in 3D as affine functions in $(x_o, y_o, z_o)$. For example $x_o(x_o, y_o, z_o) = x_o$
- For this reason, the default semantics of OpenGL is to interpolate all varying variables as functions over triangles that are affine in $(x_o, y_o, z_o)$.
- As we will see this is equivalent to a function being affine over eye coordinates, $(x_e, y_e, z_e)$, but it is not equivalent to a function being affine over normalized device coordinates, $(x_n, y_n, z_n)$ or window coordinates.

**projecting down**

- If we have a function $f$ that is affine in $(x, y, z)$ when restricted to a triangle in 3D
- then we can use the fact that the triangle is flat to write $f$ as a function that is affine in only two variables.
- idea: write $z$ as an affine function of $(x, y)$ plug this into the affine expression for $f$.
- so $f$ is also affine in $(x, y)$.

**Going Sideways**

- Suppose we have some matrix expression of the form
  \[
  \begin{bmatrix}
  x' \\
  y' \\
  z' \\
  w'
  \end{bmatrix} = P \begin{bmatrix}
  x \\
  y \\
  z \\
  1
  \end{bmatrix}
  \]  
  (3)

  for some invertible 4-by-4 matrix $P$ (where $P$ does not even have to be an affine matrix).
- Then, just looking at the four rows independently, we see that $x'$, $y'$, $z'$, and $w'$ are all affine functions of $(x, y, z)$.  

**more sideways**
If we have a function $f$ which is affine in $(x, y, z)$ then we can see that $f$ is also affine in $(x', y', z', w')$. To see this, note that:

$$
f = \begin{bmatrix} a & b & c & d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (4)
$$

$$
= \begin{bmatrix} a & b & c & d \end{bmatrix} P^{-1} \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \quad (5)
$$

$$
= \begin{bmatrix} e & g & h & i \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} \quad (6)
$$

so the property of affine-ness is in agreement between object or eye or clip coordinates, and also between normalized and window coordinates.

not sideways

• The only time we have to be careful is when division is done.
• For example, given the relation

$$
\begin{bmatrix} x'w' \\ y'w' \\ z'w' \\ w' \end{bmatrix} = P \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$

it will generally not be the case that a function $f$ which was affine in $(x, y, z)$ will be affine in $(x', y', z')$ or $(x', y', z', w')$.
• our openGL varying variables “fields” over a triangle, are not affine functions in NDC or window coordinates.

how to evaluate the varying variables

• Recall

$$
\begin{bmatrix} x_nw_n \\ y_nw_n \\ z_nw_n \\ w_n \end{bmatrix} = PM \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}
$$

• Inverting our matrices, this implies that at each point on the triangle, we have the relation:

$$
\begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix} = M^{-1}P^{-1} \begin{bmatrix} x_nw_n \\ y_nw_n \\ z_nw_n \\ w_n \end{bmatrix}
$$

..and

• A varying variable, $v$, is an affine functions of $(x_o, y_o, z_o)$
• We also make use of the obvious fact that the constant function 1 is also affine in $(x_o, y_o, z_o)$.
• Thus, for some $(a, b, c, d)$, we have

$$
\begin{bmatrix} v \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}
$$

3
and therefore:

\[
\begin{bmatrix}
  v \\
  1
\end{bmatrix}
= \begin{bmatrix}
  a & b & c & d \\
  0 & 0 & 0 & 1
\end{bmatrix}
M^{-1}P^{-1}
\begin{bmatrix}
  x_n w_n \\
  y_n w_n \\
  z_n w_n \\
  w_n
\end{bmatrix}
= \begin{bmatrix}
  e & f & g & h \\
  i & j & k & l
\end{bmatrix}
\begin{bmatrix}
  x_n \\
  y_n \\
  z_n \\
  1
\end{bmatrix}
\]

for some appropriate values e...l.

..and

- Now divide both sides by \(w_n\) and we get

\[
\begin{bmatrix}
  v / w_n \\
  1 / w_n
\end{bmatrix}
= \begin{bmatrix}
  e & f & g & h \\
  i & j & k & l
\end{bmatrix}
\begin{bmatrix}
  x_n \\
  y_n \\
  z_n \\
  1
\end{bmatrix}
\]

- conclusion: \(v / w_n\) and \(1 / w_n\) are affine functions of normalized device coordinates.

and

- “going sideways” we deduce that \(v / w_n\) and \(1 / w_n\) are affine functions of window coordinates \((x_w, y_w, z_w)\).
- “projecting down” we can conclude: \(v / w_n\) and \(1 / w_n\) are both affine functions of \((x_w, y_w)\).
- the above derivation can now be thrown away
- if we have the values of \(v / w_n\) and \(1 / w_n\) at the three vertices, we can linearly interpolate to get their values at each pixel, and then we can do a divide (per pixel) to obtain the value of \(v\) at that pixel!
- the end to end process going from \(v\) at the vertices to \(v\) at the pixels will be called “rational linear interpolation”.

in **OpenGL**

- Now we can see how OpenGL can perform “rational linear interpolation”.
  - The vertex shader is run on each vertex, calculating clip coordinates and varying variables for each vertex.
  - Clipping is run on each triangle; (This may create new vertices. Linear interpolation in clip coordinate space is run to determine the clip coordinates and varying variable values for each such new vertex.)
  - For each vertex, and for each varying variable \(v\), OpenGL creates an internal variable \(f := v / w_n\). Additionally, for each vertex OpenGL creates one internal variable \(g := 1 / w_n\).
  - For each vertex, division is done to obtain the normalized device coordinates. \(x_n = x_w / w_n\), \(y_n = y_w / w_n\), \(z_n = z_w / w_n\).
  - For each vertex, the normalized device coordinates are transformed to window coordinates.
  - The \([x_w, y_w]^T\) coordinates are used to position the triangle on the screen.
  - For every interior pixel of the triangle, linear interpolation is used to obtain the interpolated values of \(z_w, f\) (one \(f\) per varying variable) and \(g\)
  - At each pixel, the interpolated \(z_w\) value is used for z-buffering.
  - At each pixel, and for all varying variables, division is done on the interpolated internal variables to obtain the correct answer \(v = f / g\).
  - The varying variable \(v\) is passed into the fragment shader.