

affine interpolation

- we say a function f is affine in variables x, y iff
- $f = ax + by + c$ for some (a, b, c) .
 - linear + constant term
- if i have the values of an affine function f at 3 points in the $[x, y]$ plane, then the function is determined

affine interpolation is fast

- suppose f is affine in x_n, y_n
- suppose i give you values at 3 verts
- we can do linear interpolation to evaluate this affine function at every pixel
- as i move between pixels on a grid i move a constant $[dx_n, dy_n]^t$
- and f changes by a fixed amount
- the fundamental native ability of the fixed function rasterizer will be to evaluate affine functions of x_n, y_n .

z-coords

- triangles are flat
- so z_o is an affine function of x_o, y_o
- projective transforms preserve flatness of objects
- so z_n is an affine function of x_n, y_n

attribute variables

- may associate data with each vertex, say a color
- what should the color be at a point inside the triangle?
- we would like each point on the triangle to have a fixed color.
- and not change with viewpoint
- so it should be a function of x_o, y_o, z_o
- lets make color in a triangle an affine function of x_o, y_o, z_o
- note: since a triangle is flat, z_o is an affine function of x_o, y_o
- so this makes color an affine function of x_o, y_o .

Texture mapping

- lots of triangles are needed for geometric complexity
- lots of triangles are overkill for photometric complexity
 - picture of a squirrel hanging on the wall
 - if we used lots of little triangles, we would have to transform many vertices and setup many triangles
- useful solution is texture mapping.
 - texture: a discrete image
 - mapping: glue it onto a triangle
 - use the texture during triangle rendering.

texture coordinate interpolation

- we can specify the gluing by telling us the texture coordinates $[x_t, y_t]^t$ at the triangle vertices
 - note: OpenGL uses [0..1] range to address the texture image.
- when we texture map, our intention is to “glue” the texture onto the triangle onto as some 2d affine transform (rotation, translation, scale, ?skew).

$$\begin{aligned}x_t &= ax_o + by_o + c \\y_t &= dx_o + ey_o + f\end{aligned}$$

- so each of x_t and y_t are affine functions of x_o, y_o .

what to do with affine functions of x_o, y_o

- we want to interpret all of our varying variables v as affine functions of object coordinates
- but the native power of our hardware is to evaluate affine functions of normalized device coords.
- this would be wrong
 - no foreshortening in textures

what to do

- by definition there must exist some a, b, c with the property

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

derivation the correct calculation

- Given our projection matrix \mathbf{P} , recall that

$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix} = \mathbf{PM} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

And so

$$\begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix} = \mathbf{M}^{-1}\mathbf{P}^{-1} \begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix}$$

derivation ..

- putting these two together, we get

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

- now divide by w_n

$$\begin{aligned} \begin{bmatrix} \frac{v}{w_n} \\ \frac{1}{w_n} \end{bmatrix} &= \begin{bmatrix} d & e & f & g \\ h & i & j & k \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} l & m & 0 & n \\ p & q & 0 & s \end{bmatrix} \begin{bmatrix} x_n \\ y_n \\ z_n \\ 1 \end{bmatrix} \end{aligned}$$

observe

- conclusion, both v/w_n and $1/w_n$ are affine function of $[x_n, y_n]^t$
- so this data can be interpolated in the hardware using affine interpolation.
- once we have $1/w_n$ and v/w_n at a pixel, we can do a division to obtain v

how this is used in the pipeline

- at the end of the vertex shader, compute clip coordinates of each point and put it in `gl_position`.

$$\begin{bmatrix} x_n w_n \\ y_n w_n \\ z_n w_n \\ w_n \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \\ z_c \\ w_c \end{bmatrix} = \mathbf{PM} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

- the vertex shader also outputs other varying variables, such as x_t and y_t with the vertex.
- three vertices of one triangle are grabbed by the primitive assembler.
- the values $1/w_n$, x_t/w_n , and y_t/w_n are computed for each vertex.
 - same for, say, r/w_n , g/w_n , b/w_n .
 - same for ALL varying variables
- it then divides the clip coordinates by w_n to obtain normalized device coordinates for each vertex.

cont..

- the rasterizer now takes over and finds all of the pixels that are inside a triangle
- at each pixel location x_n, y_n , it computes the z_n value using affine interpolation.
- at each pixel location x_n, y_n , it also computes the $1/w_n$, x_t/w_n , y_t/w_n , r/w_n , g/w_n , b/w_n values using affine interpolation.
 - note: the rasterizer never needs to actually know about any of the matrices we used in our derivation.
- at each pixel, it then divides each x_t/w_n , y_t/w_n , r/w_n , g/w_n , b/w_n by $1/w_n$ to obtain x_t , y_t , r , g , b ...
- this data is sent to the pixel shader.
- since the hardware does this, its *effective* power is to compute affine interpolation wrt object coordinates!
- we also call this perspective correct interpolation, or linear rational interpolation

what if

- what if you set `glPosition` to be ndc's instead of clip coords?
- the triangle will be in the right place, with the right depth values as well
- the hardware will try to do affine interp wrt object coords
- but it will think w_n is 1

- so it will effectively do affine interp wrt ndc's

your assignment

- we will give you a program that texture maps two triangles and lets you spin them around in space.
- texture coordinates are passed to the vertex shader, which just passes these through to the pixel shader.
- in the vertex shader, you typically set glPosition to be the clip coordinates
- in the pixel shader, the lin-rat interpolated texture coordinates appear, using the perspective correct interpolation we just learned.

but

- you will break your vertex shader by setting glPos to be ndc's instead of clip coordinates.
- you will add stuff to your v/f shaders so that you can get the same result as above, even though the hardware is only exposing affine interp wrt ndcs.

Projector texture mapping

- suppose i want to model a slide projector projecting an image onto a triangle.
- common use: mapping real photograph of a plane onto a model of the plane
- common use: shadow mapping
- this is a sort of “gluing” but it is not an affine gluing
- model the slide projector as a pin hole model

$$\begin{bmatrix} x_t & w_t \\ y_t & w_t \\ - & \\ w_t & \end{bmatrix} = \mathbf{Q} \begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix}$$

- \mathbf{Q} can be loaded into the vertex shader, and the above multiply can be done in the vertex shader.
- question: given this desired definition for x_t and y_t at each point in a triangle..
- given that the API effectively interpolates varying variables as functions that are affine in ndcs
- what should we make our varying variables?
- what should we do with these variables in the pixel shader