Interpolation and Basis Fns

Topics

Today
- Interpolation
  - Linear and bilinear interpolation
  - Barycentric interpolation
- Basis functions
  - Square, triangle, ...
- Morphing

Thursday
- Splines and curves
  - Hermite and Catmull-Rom splines
  - Bezier curves
Let's say we have points $y_1$ and $y_2$ and we want to move smoothly between the two. One way to do this is via the line connecting the two points.
**Linear Interpolation - lerp**

Very useful function in computer graphics

```cpp
lerp(t, v0, v1) {
    return (1-t)*v0 + t*v1;
}

lerp(t, v0, v1) {
    return v0 + t*(v1-v0);
}
```

Intuitively: further from the control point, alpha is smaller.

Here’s a connection between linear interpolation and barycentric coordinates. As you move along the line...

```
m1 + m2 != 0
```

Can scale m1 and m2 by the same amount and it still works. Like barycentric coordinates.
Note that $d_1 + d_2$ sum to 1.

$m_1 + m_2 \neq 0$
Can scale $m_1$ and $m_2$ by the same amount and it still works. Like barycentric coordinates.

talk about balance point of the triangle

negative mass?
Make a slide that shows that you can interpolate anything using this formula: color, alpha, z, texcoords
Can show this visually
Discontinuous normals across verts and edges.

Shading is a function of normal; so pixels on piecewise flat regions will all receive the same color.

Looks faceted.

Physical coordinates don't change.

Shading uses interpolated normal rather than physical normal (jaggies on edge.)
instead of color, normal interpolation, can also interpolate tex coords.

Interpolate texture coordinates over triangle

Texture + texture coordinates creates texture mapped image

Don’t show texture and triangle aligned like this

\[(u,v) = \text{interpolate}(x,y,v_0,v_1,v_2)\]

Differences between \((u, v)\) and \((s, t)\)

--

For each pixel \((x, y)\)

\[s,t = \text{texmap}(\alpha_0, \alpha_1, \alpha_2)\]

\[c = \text{texture}(s,t)\]

Given these weights and the texture coordinates at the vertices, compute the texture coordinates \((s,t)\) at the interior pixel.
Map Texture Image onto Triangles

1. Specify texture coordinates at triangle vertices \((s_i, t_i)\)

Map Texture Image onto Triangles
you could also use the closest vertex (nearest neighbor)
Map Texture Image onto Triangles

3. Copy the color from the texture at \((s, t)\) into that pixel

\[
(s, t) = \text{texmap}(\alpha_0, \alpha_1, \alpha_2) = \alpha_0(s_0, t_0) + \alpha_1(s_1, t_1) + \alpha_2(s_2, t_2)
\]

The texture space is also quantized; (texels rather than pixels.)

You might have to do more interpolation to get the final color.

You could just use the color of the nearest texel.
(what if density of texels is much greater than pixels? what if much smaller?)

Of course usually we’re not thinking about applying textures to just a single triangle. We might apply them to a 3D mesh or window.

Texture are defined just like windows
Bilinear Interpolation

Bilinear Interpolation
This is the formula for interpolating one texel show the demo!

Can anyone guess what type of ruled surface this forms? (kb - I think it’s a hyperbolic paraboloid.)

Ruled surface:
\[ S(u,v) = v \ a(u) + (1-v) \ b(u) \]
Basis Functions
Let's revisit linear interpolation.
Another way to think about linear interpolation is as the construction of a surface (even 1d “surface”) via basis functions.
Here the functions are “hat” or triangle functions with height $y_i$ and width of two units.

can you imagine what it would look like if the sum of the two functions went over or under the line between $y_1$ and $y_2$? would you have a linear interpolation then?

Work through derivation of linear interpolation formula

Define $T_1 = T(t-1)$, $T_2 = T(t-2)$, etc.
this is what we saw in the slides on texture mapping.
In general, an interpolating function can be anything where...

**Basis Functions**

**Basic formula**

\[ y(t) = \sum_{i=0}^{n} y_i B_i(t) \]

**Basis functions**

\[ B_i(t) \]

*Often i’th functions are shifted versions of 0’th*
*Basis functions are like x, y, and z basis vectors*

**Interpolating Function**

**Necessary conditions:**

\[ B_i(0) = 1 \]

\[ B_i(k) = 0 \]

*True for triangle and square basis functions*
Feature-Based Metamorphosis

Simultaneous Warp and Cross-Dissolve
Beier and Neely, SIGGRAPH 1992

Show Black and White video

http://www.hammerhead.com/thad/morph.html

http://youtu.be/F2AitTPl5U0 (see 5:15)

relationship between points and features stays “same”
Should we scale the F uniformly?
what do you do if you have two lines?
what do you do if you have two lines?
Combine the two into a one-to-one mapping.

The weights should sum to 1

One way to do this is to weight by inverse distance.

\[ w_i = \left( \frac{\text{len}^p(\mathbf{e}_i)}{a + \text{dist}(\mathbf{p}, \mathbf{e}_i)} \right)^b \]

Show normalize the weights

Points on the edge should not be influenced by other points. This implies the weight should increase the closer we are to the edge.
Outline this better
We have a list of edges that define the features on the first image
We have a list of edges that define the features on the second image
How do we define an intermediate image?

First, we interpolate the features between the two images
The features are shown on the right
Morphing

Five different uses of interpolation!

1. Interpolate in time to control pace
2. Interpolate line segments to define the warp
3. Line segments define a spatial warp
4. Bilinearly interpolate images when warping
5. Interpolate colors to cross-dissolve

Things to Remember

Interpolation
- Widely used in graphics: image, texture, curves and surfaces, animation
- Nearest neighbor, linear, and bilinear

Basis functions
- Square
- Triangle
- Many others: cubic, trig, sinc, wavelets, ...

Feature-based interpolation
- Create basis functions that depend on distance