Review of Tuesday

- We have learnt how to rasterize lines and fill polygons
- Colors (and other attributes) are specified at vertices
- Interpolation required to fill polygon with attributes
Review of Tuesday

• Mathematical formulation:

Given \((x_i, f_i)\) ("positions" and "values")

Find function \(f\) defined on \([x_0, x_n]\)

such that \(f(x_i) = f_i\)

• We required our solutions to be linear functions
Interpolation: Linear

- Linear interpolation

Parametric function takes the form

\[ f(u) = (1 - u)f_0 + uf_1 \]

Values between \( u_0 \) and \( u_1 \) are “mixtures” of \( f_0 \) and \( f_1 \)

\[ u = 0.2 \quad \rightarrow \quad 80\% f_0 + 20\% f_1 \]

Control points are weighted/blended together
Interpolation: Linear

- Basis functions/Blending functions/Weights

\[ f(u) = (1 - u)f_0 + uf_1 \]

\[ (1 - u) + u = 1 \quad \text{partition of unity} \]

\[ f(u) = b_0(u) \cdot f_0 + b_1(u) \cdot f_1 \]

\[ f(u) = \sum_{i} b_i(u) \cdot f_i \quad \text{control point contributions are blended together} \]
What about colors?

- Linear interpolation (component-wise)

\[ \mathbf{f}_{RGB}(t) = (f^R(t), f^G(t), f^B(t)) \]
What about colors?

- Linear interpolation across triangle
- Can we construct function as mixture of corner vertices?

\[ f(\alpha_0, \alpha_1, \alpha_2) = \alpha_0 \cdot f_0 + \alpha_1 \cdot f_1 + \alpha_2 \cdot f_2 \]
What about colors?

- **Barycentric interpolation** across triangle

\[ p = \alpha_0 p_0 + \alpha_1 p_1 + \alpha_2 p_2 \quad \text{Compute weights (barycentric coordinates)} \]

\[ f(p) = \alpha_0 f_0 + \alpha_1 f_1 + \alpha_2 f_2 \quad \text{Interpolate values} \]

\[ \alpha_0 + \alpha_1 + \alpha_2 = 1 \quad \text{convex combination} \]

\[ 0 \leq \alpha_0, \alpha_1, \alpha_2 \leq 1 \]
What about colors?
What about colors?

- Polygon interpolation is dependent on triangulation
How about patterns?

- Define color on more positions than just vertices?
- Map an image onto a mesh – add details
Texture Mapping

- 2D texture coordinates \((s, t)\) specified per vertex

- Interpolation of coordinates takes care of the rest
Texture Mapping and Rasterization

• How do we map texels to pixels?

• Aliasing
  • Look up texel at pixel center (point sampling; nearest neighbor)

• Linear filtering (average texels around pixel center location)
Texture Mapping and Rasterization

- **Magnification**: texel maps to multiple pixels
- **Minification**: pixel maps to multiple texels

![Image showing texture mapping examples]

OpenGL demo in class
Texture Mapping and Rasterization

- **Mip-Mapping:**
  - Create low resolution texture images for minification
  - Similar to low-pass filtering before sampling
  - Faster rendering
  - Reduced aliasing
Textures on curved 3D objects are distorted
Triangles and Rasterization - Summary

• Summary:
  • Vertices, lines, triangles are easily mapped to fragments
  • Rasterization produces fragments from continuous primitives
  • Fragments have interpolated attributes
    • Color
    • Texture coordinates
    • Alpha
  • Strong aliasing effects can be caused by sampling
General Object Representations

• So far we have described objects as (triangle) meshes
• Triangle meshes are piecewise linear object representations
  • Easy to rasterize/“put on the screen”
  • Maps nicely to graphics pipeline

• What about other representation techniques?
• Disconnect rendering from object representation?
• Editing?
• Vector graphics?
General Object Representations

- Three ways to represent curves and surfaces

- Explicit
  \[ y = f(x) \]

- Implicit
  \[ f(x, y) = 0 \]

- Parametric
  \[ f(u) = (x(u), y(u), z(u)) \]
General Object Representations

- Explicit representation

\[ y = f(x) \]

*Independent variable* defines value of *dependent variable*
General Object Representations

- Implicit representation

\[ f(x, y) = 0 \]

Points in space can be tested for membership

In 2D these functions define curves, in 3D they define surfaces.
General Object Representations

- Parametric representation

\[ f(u) = (x(u), y(u), z(u)) \]

Coordinates expressed in terms of one independent variable, the parameter.
Parametric Representation

- Set of **control points**
- Piecewise linear representation vs. higher-order parametric curves

![Diagram showing linear representation and Bézier curve (cubic) approximation]

- Linear representation
- Linear interpolation
- Bézier curve (cubic)
- Approximation
Bézier Curves

- Relaxing the interpolation condition allows for more stable curve construction (approximation)
- Goals:
  - Easy construction
  - Suitable for rendering and basic geometric operations
  - Control over smoothness
  - Stability
  - Easy computation of derivatives
Bézier Curves

- Geometric construction of a cubic Bézier curve

Four control points are weighted to construct a cubic Bézier curve.
Bézier Curves

- Bézier curve constructed by repeated linear interpolation

\[ f(u) = \sum_{i=0}^{n} b_{in}(u) p_i \]

curve blends together control points

**cubic Bernstein basis polynomials**

\[ b_{in}(u) = \binom{n}{i} u^i (1 - u)^{n-i} \]
Bézier Curves

• Convex hull property

\[ b_{in}(u) = \binom{n}{i} u^i (1 - u)^{n-i} \]

\[ \sum_{i=0}^{n} b_{in}(u) = 1 \quad \text{partition of unity} \quad 0 \leq b_{in}(u) \leq 1 \]

Bézier polynomial is convex combination of control points; stays Within convex hull of control polygon.
Bézier Curves

- **Subdivision** – example: cubic Bézier curve

\[ p^k_i(u) = (1 - u) \cdot p^k_{i-1} + u \cdot p^k_{i-1} \]

**cubic Bézier curve**

\[ f(u) = p^3_3(u) \]

\( (p_0^0, p_1^1, p_2^2, p_3^3) \)  
**Cubic curve 1**

\( (p_3^3, p_2^3, p_1^3, p_0^3) \)  
**Cubic curve 2**
Bézier Curves

• Derivatives

\[ f(u) = \sum_{i=0}^{n} b_{in}(u) \cdot p_i \]

Derivatives are tangents. Lighting computations require derivatives. Example: Utah Teapot consists of bicubic Bézier patches
Bézier Curves

• Derivatives

\[ f(u) = \sum_{i=0}^{n} b_{in}(u) \cdot p_i \]

\[ \frac{df(u)}{du} = \frac{d}{du} \sum_{i=0}^{n} b_{in}(u) \cdot p_i = \sum_{i=0}^{n} \frac{db_{in}(u)}{du} \cdot p_i \]

\[ \frac{db_{in}(u)}{du} = n(b_{i-1,n-1}(u) - b_{i,n-1}(u)) \]

\[ \frac{df(u)}{du} = \sum_{i=0}^{n} n(b_{i-1,n-1}(u) - b_{i,n-1}(u)) \cdot p_i \]
Bézier Curves

- Derivatives

\[
\frac{df(u)}{du} = \sum_{i=0}^{n} n(b_{i-1,n-1}(u) - b_{i,n-1}(u)) \cdot p_i
\]

\[
\frac{df(u)}{du} = \sum_{i=0}^{n-1} b_{i,n-1}(u) \cdot n(p_{i+1} - p_i)
\]

Derivative is a Bézier curve of order \((n-1)\) with “combined” control points
Bézier Curves

- **Joining** Bézier curves

Discontinuous

- $C^0$
  - $f$ continuous
  - $f'$ discontinuous

- $C^1$
  - $f$ continuous
  - $f'$ continuous