Outline

• Math review
• Introduction to 2D curves
• Functional representations
• Parametric cubic curves
• Introduction to Bézier curves
• Continuity
Geometric Preliminaries

• **Affine Geometry**
  – Scalars + Points + Vectors and their ops

• **Euclidean Geometry**
  – Affine Geometry lacks angles, distance
  – New op: Inner/Dot product, which gives
    • Length, distance, normalization
    • Angle, Orthogonality, Orthogonal projection

• **Projective Geometry**
Affine Geometry

• **Affine Operations:**

  ![Diagram](affine_operations)

  - vector $\leftarrow$ scalar $\cdot$ vector,
  - vector $\leftarrow$ vector $/$ scalar
  - vector $\leftarrow$ vector $+$ vector,
  - vector $\leftarrow$ vector $-$ vector
  - vector $\leftarrow$ point $-$ point
  - point $\leftarrow$ point $+$ vector,
  - point $\leftarrow$ point $-$ vector

  scalar-vector multiplication
  vector-vector addition
  point-point difference
  point-vector addition

• **Affine Combinations:** $\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$
  where $v_1, v_2, \ldots, v_n$ are vectors and $\Sigma_i \alpha_i = 1$

  **Example:** $R = (1 - \alpha)P + \alpha Q$

  ![Diagram](affine_combinations)
Mathematical Preliminaries

- Vector: an $n$-tuple of real numbers
- Vector Operations
  - Vector addition: $u + v = w$
    - Commutative, associative, identity element (0)
  - Scalar multiplication: $cv$

- Note: Vectors and Points are different
  - Can not add points
  - Can find the vector between two points
Linear Combinations & Dot Products

• A *linear combination* of the vectors $v_1, v_2, \ldots v_n$
  is any vector of the form
  $\alpha_1 v_1 + \alpha_2 v_2 + \ldots + \alpha_n v_n$
  where $\alpha_i$ is a real number (i.e. a scalar)

• *Dot Product*:

$$u \cdot v = \sum_{k=1}^{n} u_k v_k$$

a real value $u_1 v_1 + u_2 v_2 + \ldots + u_n v_n$ written as $u \cdot v$
Fun with Dot Products

• *Euclidian Distance* from \((x,y)\) to \((0,0)\):

\[
\sqrt{x^2 + y^2}
\]

in general:

\[
\sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}
\]

which is just:

\[
\sqrt{\vec{x} \cdot \vec{x}}
\]

• This is also the length of vector \(\vec{v}\):

\[
\|\vec{v}\| \text{ or } |\vec{v}|
\]

• *Normalization* of a vector: \(\hat{\vec{v}} = \frac{\vec{v}}{|\vec{v}|}\).

• *Orthogonal* vectors: \(\vec{u} \cdot \vec{v} = 0\)
Projections & Angles

- **Angle between vectors, $\theta$**
  
  $\vec{u} \cdot \vec{v} = |\vec{u}| |\vec{v}| \cos(\theta)$

  
  $\theta = \text{ang}(\vec{u}, \vec{v}) = \cos^{-1} \left( \frac{\vec{u} \cdot \vec{v}}{|\vec{u}| |\vec{v}|} \right) = \cos^{-1}(\hat{u} \cdot \hat{v})$.

- **Projection of vectors**

  $\vec{u}_1 = \frac{(\vec{u} \cdot \vec{v})}{(\vec{v} \cdot \vec{v})} \vec{v}$
  
  $\vec{u}_2 = \vec{u} - \vec{u}_1$. 

Pics/Math courtesy of Dave Mount @ UMD-CP
Matrices and Matrix Operators

- A $n$-dimensional vector:

- Matrix Operations:
  - Addition/Subtraction
  - Identity
  - Multiplication
    - Scalar
    - Matrix Multiplication

- Implementation issue:
  Where does the index start?
  (0 or 1, it’s up to you…)

\[
\begin{bmatrix}
  x_1 \\
  \cdot \\
  \cdot \\
  \cdot \\
  x_n
\end{bmatrix}
\]

\[
A + B = B + A \\
A + (B + C) = (A + B) + C \\
(cd)A = c(dA) \\
1A = A \\
c(A + B) = cA + cB \\
(c + d)A = cA + dA
\]
Matrix Multiplication

• \([C] = [A][B]\)
• Sum over rows & columns
• Recall: matrix multiplication is *not* commutative

*Identity Matrix:*
1s on diagonal \(c_{ij} = \sum_{s=1}^{m} a_{is} b_{sj}\)
0s everywhere else
Matrix Determinants

• A single real number
• Computed recursively \( \det(A) = \sum_{j=1}^{n} A_{i,j} (-1)^{i+j} M_{i,j} \)
• Example:
  \[
  \begin{vmatrix}
    a & c \\
    b & d \\
  \end{vmatrix}
  = ad - bc
  
  \]
• Uses:
  – Find vector ortho to two other vectors
  – Determine the plane of a polygon
Cross Product

• Given two non-parallel vectors, $A$ and $B$
• $A \times B$ calculates third vector $C$ that is orthogonal to $A$ and $B$
• $A \times B = (a_y b_z - a_z b_y, a_z b_x - a_x b_z, a_x b_y - a_y b_x)$

$$A \times B = \begin{vmatrix} \vec{x} & \vec{y} & \vec{z} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$
Matrix Transpose & Inverse

- **Matrix Transpose**: Swap rows and cols:
  \[ A = \begin{bmatrix} 2 \\ 8 \end{bmatrix}, \quad A^T = \begin{bmatrix} 2 & 8 \end{bmatrix} \]

- Facts about the transpose:
  \[(A^T)^T = A\]
  \[(A + B)^T = A^T + B^T\]
  \[(cA)^T = c(A^T)\]
  \[(AB)^T = B^T A^T\]

- **Matrix Inverse**: Given \(A\), find \(B\) such that
  \[AB = BA = I, \quad B \rightarrow A^{-1}\]

(only defined for square matrices)
Derivatives of Polynomials

\[ f(x) = \alpha x^n \]

\[ \frac{df(x)}{dx} = \alpha nx^{n-1} \]

\[ f(x) = 5x^3 \]

\[ \frac{df(x)}{dx} = 15x^2 \]
Partial Derivatives of Polynomials

\[ f(x, y) = \alpha x^n y^m \]
\[ \frac{\partial f(x, y)}{\partial x} = \alpha nx^{n-1} y^m \]
\[ f(x, y) = 5x^3 y^4 \]
\[ \frac{\partial f(x, y)}{\partial x} = 15x^2 y^4 \]
Curves
Example Application: Font Design and Display

• Curved objects are everywhere

• There is always need for:
  – mathematical fidelity
  – high precision
  – artistic freedom and flexibility
  – physical realism

Pics/Math courtesy of G. Farin @ ASU
Example Application: Graphic Design and Arts

Courtesy of http://www.pilot3d.com
Example Application: Tool Path Generation
Example Application: Motion Planning
Functional Representations

• *Explicit Functions*:  
  – representing one variable with another  
  – fine if $\exists$ only one $x$ value for each $y$ value  
  – Problem: what if I have a sphere? 

\[ z = \pm \sqrt{r^2 - x^2 - y^2} \]

• Multiple values …. (not used in graphics)
Functional Representations

• **Implicit Functions:**
  – curves/surfaces represented as “the zeros”
  – good for rep. of \((n-1)D\) objects in \(nD\) space
  – Sphere example: \(x^2 + y^2 + z^2 - r^2 = 0\)
  – What class of function?
    • *polynomial*: linear combo of integer powers of \(x,y,z\)
    • *algebraic curves & surfaces*: rep’d by implicit polynomial functions
    • *polynomial degree*: total sum of powers, i.e. polynomial of degree 6: \(x^2 + y^2 + z^2 - r^2 = 0\)
Functional Representations

- **Parametric Functions:**
  - 2D/3D curve: two functions of one parameter
    \[(x(u), y(u)) \quad \text{or} \quad (x(u), y(u), z(u))\]
  - 3D surface: three functions of two parameters
    \[(x(u,v), y(u,v), z(u,v))\]
  - Example: Sphere
    \[x(\theta, \phi) = \cos \phi \cos \theta\]
    \[y(\theta, \phi) = \cos \phi \sin \theta\]
    \[z(\theta, \phi) = \sin \phi\]

Note: rep. not algebraic, but is parametric.
Functional Representations

• Which is best??
  – It depends on the application
  – Implicit is good for
    • computing ray/surface intersection
    • point inclusion (inside/outside test)
    • mass & volume properties
  – Parametric is good for
    • subdivision, faceting for rendering
    • Surface & area properties
    • popular in graphics
Issues in Specifying/Designing Curves/Surfaces

• Note: the internal mathematical representation can be very complex
  – high degree polynomials
  – hard to see how parameters relate to shape

• How do we deal with this complexity?
  – Use *curve control points* and either
    • Interpolate
    • Approximate
Points to Curves

• The *Lagrangian interpolating polynomial*
  – \( n+1 \) points, the unique polynomial of degree \( n \)
  – curve wiggles thru each control point
  – Issue: not good if you want smooth or flat curves

• *Approximation* of control points
  – points are *weights* that tug on the curve or surface

![Interpolation](image1.png) ![Approximation](image2.png)

Pics/Math courtesy of Dave Mount @ UMD-CP
Warning, Warning, Warning: Pending Notation Abuse

- $t$ and $u$ are used interchangeably as a parameterization variable for functions

- Why?
  - $t$ historically is "time", certain parametric functions can describe "change over time" (e.g. motion of a camera, physics models)
  - $u$ comes from the 3D world, i.e. where two variables describe a B-spline surface
    - $u$ and $v$ are the variables for defining a surface

- Choice of $t$ or $u$ depends on the text/reference
Parametric Curves

• General rep:
  \[ x = x(t), \ y = y(t), \ z = z(t) \]

• Properties:
  – individual functions are single-valued
  – approximations are done with piecewise polynomial curves
  – Each segment is given by three cubic polynomials \((x,y,z)\) in parameter \(t\)
  – Concise representation
Cubic Parametric Curves

• Balance between
  – Complexity
  – Control
  – Wiggles
  – Amount of computation
  – Non-planar
Parametric Curves

- Cubic Polynomials that define a parametric curve segment

\[ Q(t) = [x(t) \ y(t) \ z(t)]^T \]

are of the form

\[
\begin{align*}
x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\
y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y, \\
z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z,
\end{align*}
\]

- Notice we restrict the parameter \( t \) to be \( 0 \leq t \leq 1 \).
Parametric Curves

• If *coefficients* are represented as a matrix

\[ C = \begin{bmatrix}
  a_x & b_x & c_x & d_x \\
  a_y & b_y & c_y & d_y \\
  a_z & b_z & c_z & d_z \\
\end{bmatrix} \]

and

\[ T = \begin{bmatrix}
  t^3 & t^2 & t & 1 \\
\end{bmatrix}^T \]

then:

\[ Q(t) = \begin{bmatrix}
  x(t) & y(t) & z(t) \\
\end{bmatrix}^T = C \cdot T \]
Parametric Curves

• $Q(t)$ can be defined with four constraints
  – Rewrite the coefficient matrix $C$ as $C = G \cdot M$
    where $M$ is a 4x4 \textit{basis matrix}, and $G$ is a four-element
    constraint matrix (\textit{geometry matrix})

• Expanding $Q(t) = G \cdot M \cdot T$ gives:

$$Q(t) = \begin{bmatrix} x(t) \\ y(t) \\ z(t) \end{bmatrix} = \begin{bmatrix} G_1 & G_2 & G_3 & G_4 \end{bmatrix} \begin{bmatrix} m_{11} & m_{21} & m_{31} & m_{41} \\ m_{12} & m_{22} & m_{32} & m_{42} \\ m_{13} & m_{23} & m_{33} & m_{43} \\ m_{14} & m_{24} & m_{34} & m_{44} \end{bmatrix} \begin{bmatrix} t^3 \\ t^2 \\ t \\ 1 \end{bmatrix}$$

$Q(t)$ is a weighted sum of the columns of the geometry matrix, each of which represents a point or vector in 3-space
Parametric Curves

• Multiplying out $Q(t) = G_x \cdot M \cdot T$ gives

$x(t) = (t^3m_{11} + t^2m_{21} + tm_{31} + m_{41})g_{1x} + (t^3m_{12} + t^2m_{22} + tm_{32} + m_{42})g_{2x} + (t^3m_{13} + t^2m_{23} + tm_{33} + m_{43})g_{3x} + (t^3m_{14} + t^2m_{24} + tm_{34} + m_{44})g_{4x}$

(i.e. just weighted sums of the elements)

• The weights are cubic polynomials in $t$ (called the blending functions, $B=MT$), $Q(t) = G_x \cdot B$

• $M$ and $G$ matrices vary by curve
  – Hermite, Bézier, spline, etc.
Some Types of Curves

• Hermite
  – def’ d by two end points and two tangent vectors

• Bézier
  – two end points plus two control points for the tangent vectors

• Splines
  – Basis Splines
  – def’ d w/ 4 control points
  – Uniform, nonrational $B$-splines
  – Nonuniform, nonrational $B$-splines
  – Nonuniform, rational $B$-splines ($\textit{NURBS}$)
Convex Hulls

- The smallest convex container of a set of points
- Both practically and theoretically useful in a number of applications
Bézier Curves

• Pierre Bézier @ Rénault ~1960

• Basic idea
  – four points
  – Start point $P_0$
  – End point $P_3$
  – Tangent at $P_0$, $P_0P_1$
  – Tangent at $P_3$, $P_2P_3$
Bézier Curves

An Example:

- **Geometry matrix** is
  \[ G_B = \begin{bmatrix} P_1 & P_2 & P_3 & P_4 \end{bmatrix} \]
  where \( P_i \) are control points for the curve

- **Basis Matrix** is
  \[
  M_B = \begin{bmatrix}
  -1 & 3 & -3 & 1 \\
  3 & -6 & 3 & 0 \\
  -3 & 3 & 0 & 0 \\
  1 & 0 & 0 & 0
  \end{bmatrix}
  \]
Bézier Curves

- The general representation of a Bézier curve is
  \[ Q(t) = G_B \cdot M_B \cdot T \]
  where
  - \( G_B \) - Bézier Geometry Matrix
  - \( M_B \) - Bézier Basis Matrix

  which is (multiplying out):
  \[ Q(t) = (1 - t)^3 P_1 + 3t (1 - t)^2 P_2 + 3t^2 (1 - t) P_3 + t^3 P_4 \]
Bernstein Polynomials (1911)

• The general form for the \( i \)-th Bernstein polynomial for a degree \( k \) Bézier curve is

\[
b_{ik}(u) = \binom{k}{i} (1 - u)^{k-i} u^i.
\]

• Some properties of BPs
  – Invariant under transformations
  – Form a partition of unity, i.e. summing to 1
  – Low degree BPs can be written as high degree BPs
  – BP derivatives are linear combo of BPs
  – Form a basis for space of polynomials w/ \( \text{deg} \leq k \)
Bernstein Polynomials

• For those that forget combinatorics

\[ b_{ik}(u) = \frac{k!}{i!(k - i)!} (1 - u)^{k-i} u^i \]

• Note: k does not have to be 3.
Joining Bézier Segments: The Bernstein Polynomials

• Cubic Bernstein blending functions

\[
\begin{align*}
    b_{03}(u) &= (1 - u)^3 \\
    b_{13}(u) &= 3u(1 - u)^2 \\
    b_{23}(u) &= 3u^2(1 - u) \\
    b_{33}(u) &= u^3.
\end{align*}
\]

• Observe: the coefficients are just rows in Pascal’s triangle

\[
\begin{array}{cccccc}
    & 1 & & & & \\
    & 1 & 1 & & & \\
    1 & 2 & 1 & & & \\
    1 & 3 & 3 & 1 & & \\
    1 & 4 & 6 & 4 & 1 & \\
\end{array}
\]
Joining Bézier Segments: The Bernstein Polynomials

- Observe

\[ Q(t) = (1-t)^3P_1 + 3t(1-t)^2P_2 + 3t^2(1-t)P_3 + t^3P_4 \]

The Four *Bernstein polynomials*
- also defined by \( B_B = M_B \cdot T \)

- These represent the blending proportions among the control points
Joining Bézier Segments: The Bernstein Polynomials

- The four cubic Bernstein polynomials
  \[ B_B = M_B \cdot T \]

- Observe:
  - at \( t=0 \), only \( B_{B1} \) is \( >0 \)
    - curve interpolates \( P1 \)
  - at \( t=1 \), only \( B_{B4} \) is \( >0 \)
    - curve interpolates \( P4 \)

Pics/Math courtesy of Dave Mount @ UMD-CP
General Form of Bezier Curve

\[
Q(u) = \sum_{i=0}^{k} P_{i+1} \binom{k}{i}(1 - u)^{k-i} u^i
\]

Control points: \(P_1, P_2, \ldots, P_{k+1}; \quad 0 \leq u \leq 1\)

Produces a point on curve \(Q\) at parameter value \(u\)
Properties of Bézier Curves

• $k+1$ control points defines a degree $k$ curve
• Affine invariance
• Invariance under affine parameter transformations
• Convex hull property
  – curve lies completely within original control polygon
• Endpoint interpolation
• Intuitive for design
  – curve mimics the control polygon
Issues with Bézier Curves

• Creating complex curves may (with lots of wiggles) requires many control points
  – potentially a very high-degree polynomial
• Bézier blending functions have *global support* over the whole curve
  – move just one point, change whole curve
• *Improved Idea:* link \((C^1)\) lots of low degree \((cubic)\) Bézier curves end-to-end
Programming Assignment 1

- Process command-line arguments
- Read in 3D control points
- Iterate through parameter space by du
- At each u value evaluate Bezier curve formula to produce a sequence of 3D points
- Output points by printing them to standard out as a polyline and control points as spheres in Open Inventor format