CSE 167:
Introduction to Computer Graphics
Lecture #9: Culling

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2014

Announcements

- Project 3 due Friday, Nov. 7th at 3:30pm
  - Don't forget to upload source code to Ted by the deadline!

Modifying the Scene

- Change tree structure
  - Add, delete, rearrange nodes
- Change node parameters
  - Transformation matrices
  - Shape of geometry data
  - Materials
- Create new node subclasses
  - Animation, triggered by timer events
  - Dynamic "helicopter-mounted" camera
  - Light source
- Create application dependent nodes
  - Video node
  - Web browser node
  - Video conferencing node
  - Terrain rendering node

Benefits of a Scene Graph

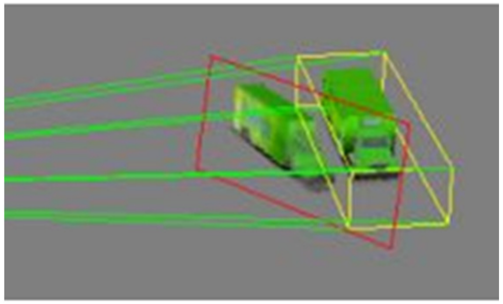
- Can speed up rendering by efficiently using low-level API
  - Avoid state changes in rendering pipeline
  - Render objects with similar properties in batches (geometry, shaders, materials)
- Change parameter once to affect all instances of an object
- Abstraction from low level graphics API
  - Easier to write code
  - Code is more compact
- Can display complex objects with simple APIs
  - Example: osgEarth class provides scene graph node which renders a Google Earth-style planet surface

Lecture Overview

- Scene Graphs & Hierarchies
  - Introduction
  - Data structures
- Performance Optimization
  - Level-of-detail techniques
  - View Frustum Culling

Level-of-Detail Techniques

- Don’t draw objects smaller than a threshold
  - Small feature culling
  - Popping artifacts
- Replace 3D objects by 2D impostors
  - Textured planes representing the objects
- Adapt triangle count to projected size



Impostor generation

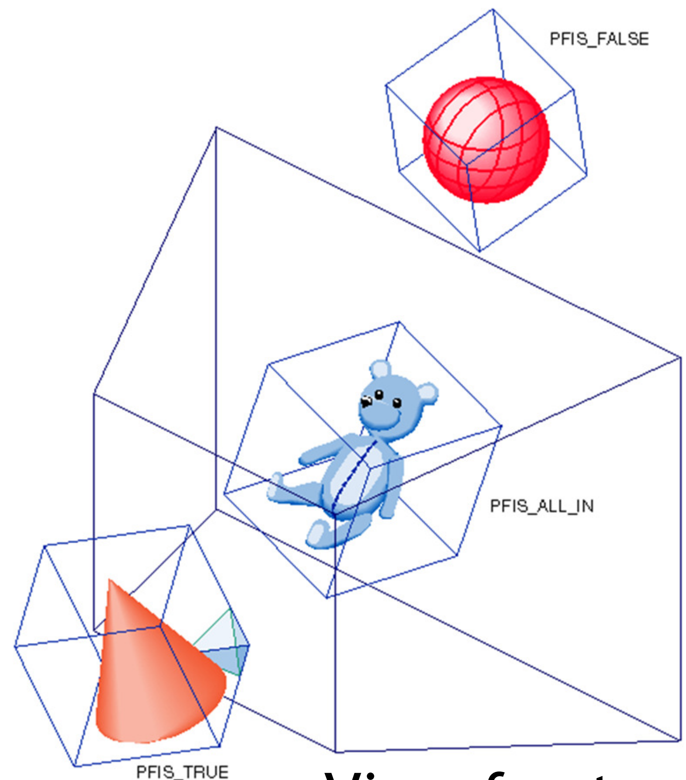
Original vs. impostor

Size dependent mesh reduction
(Data: Stanford Armadillo)

UCSD

View Frustum Culling

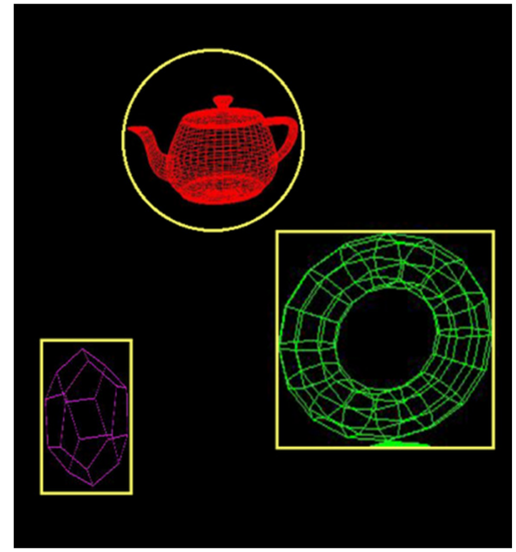
- Frustum defined by 6 planes
- Each plane divides space into "outside", "inside"
- Check each object against each plane
  - Outside, inside, intersecting
- If "outside" all planes
  - Outside the frustum
- If "inside" all planes
  - Inside the frustum
- Else partly inside and partly out
- Efficiency

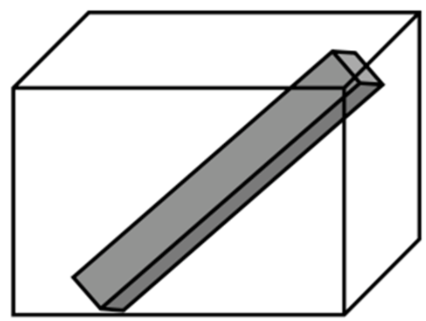


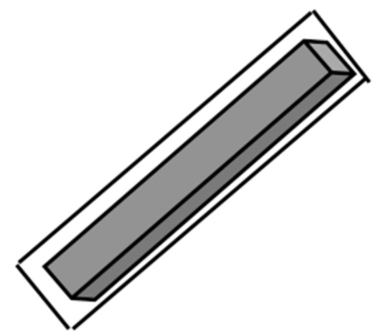
View frustum

Bounding Volumes

- Simple shape that completely encloses an object
- Generally a box or sphere
- We use spheres
  - Easiest to work with
  - But hard to calculate tight fits
- Intersect bounding volume with view frustum instead of each primitive

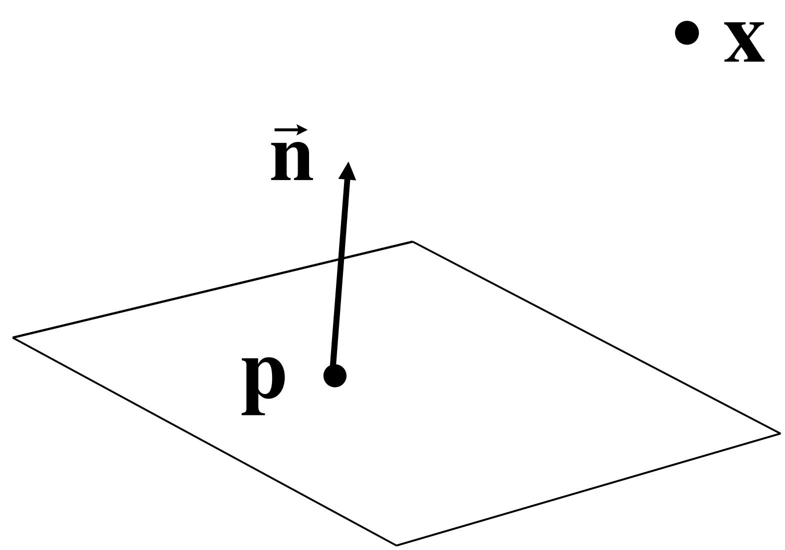






Distance to Plane

- A plane is described by a point **p** on the plane and a unit normal **n**
- Find the (perpendicular) distance from point **x** to the plane

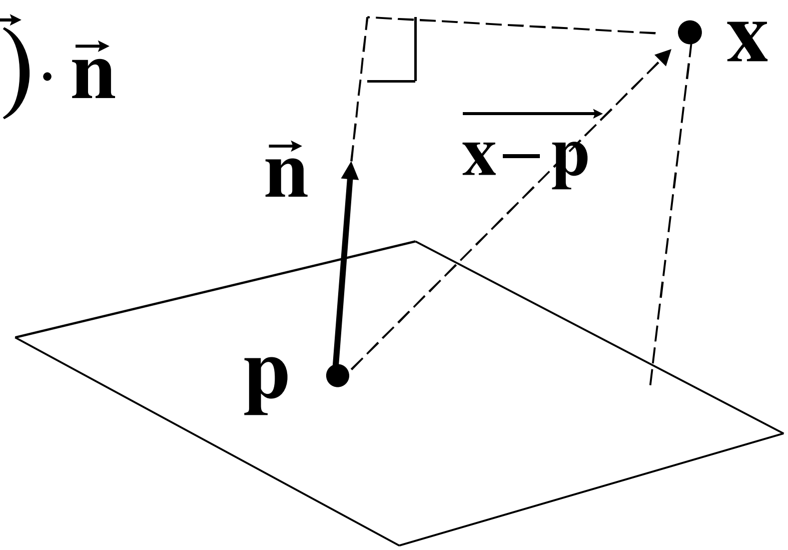


UCSD

Distance to Plane

- The distance is the length of the projection of **x-p** onto **n**

$$dist = \overrightarrow{(\mathbf{x}-\mathbf{p})} \cdot \vec{\mathbf{n}}$$

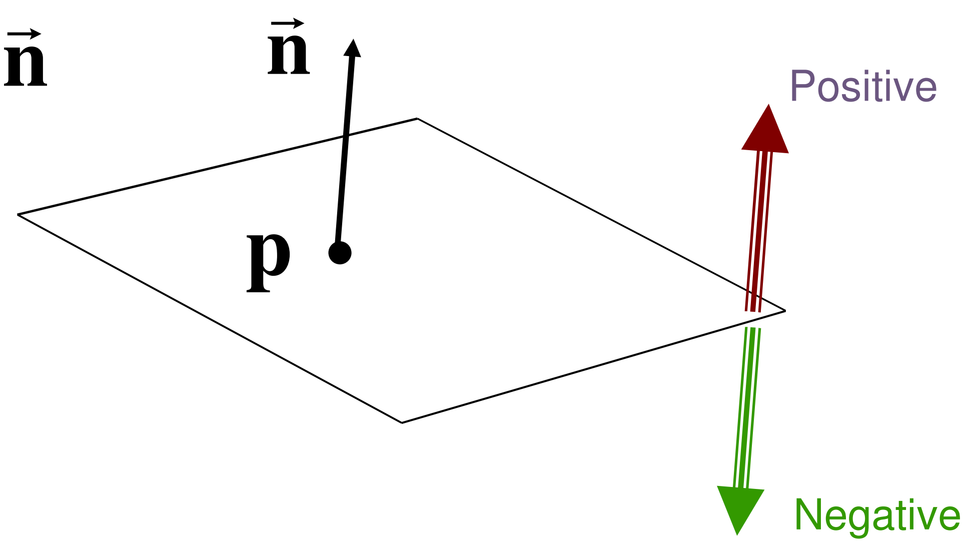


UCSD

Distance to Plane

- The distance has a sign
  - positive on the side of the plane the normal points to
  - negative on the opposite side
  - zero exactly on the plane
- Divides 3D space into two infinite half-spaces

$$dist(\mathbf{x}) = \overrightarrow{(\mathbf{x} - \mathbf{p})} \cdot \vec{\mathbf{n}}$$



Distance to Plane

- Simplification

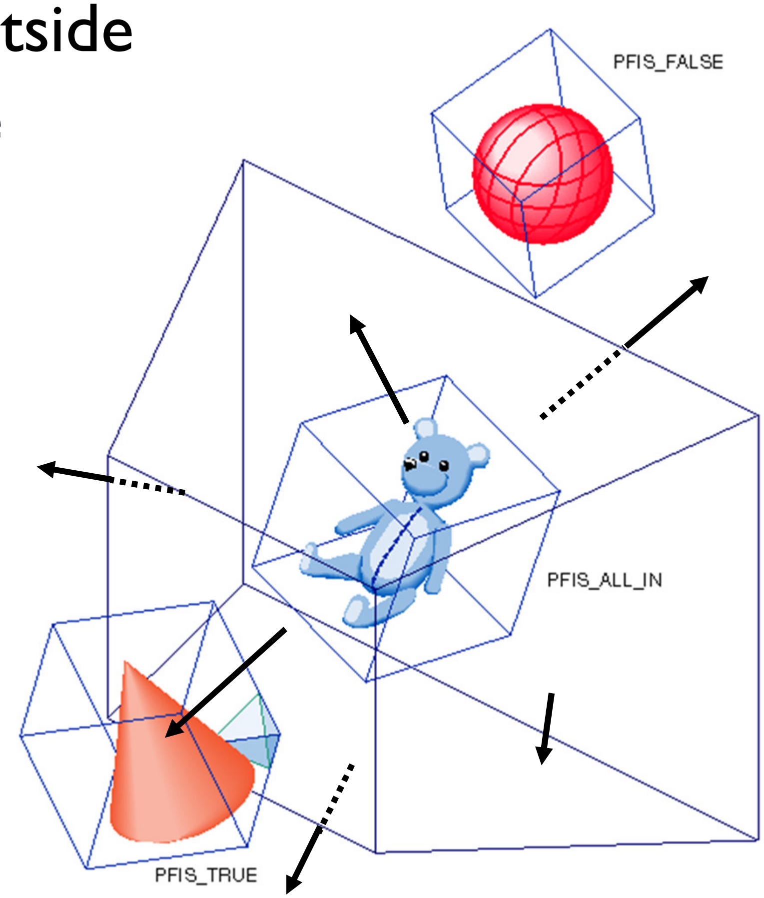
$$\begin{aligned} dist(\mathbf{x}) &= (\mathbf{x} - \mathbf{p}) \cdot \mathbf{n} \\ &= \mathbf{x} \cdot \mathbf{n} - \mathbf{p} \cdot \mathbf{n} \end{aligned}$$

$$dist(\mathbf{x}) = \mathbf{x} \cdot \mathbf{n} - d, \quad d = \mathbf{pn}$$

- *d* is independent of **x**
- *d* is distance from the origin to the plane
- We can represent a plane with just *d* and **n**

Frustum With Signed Planes

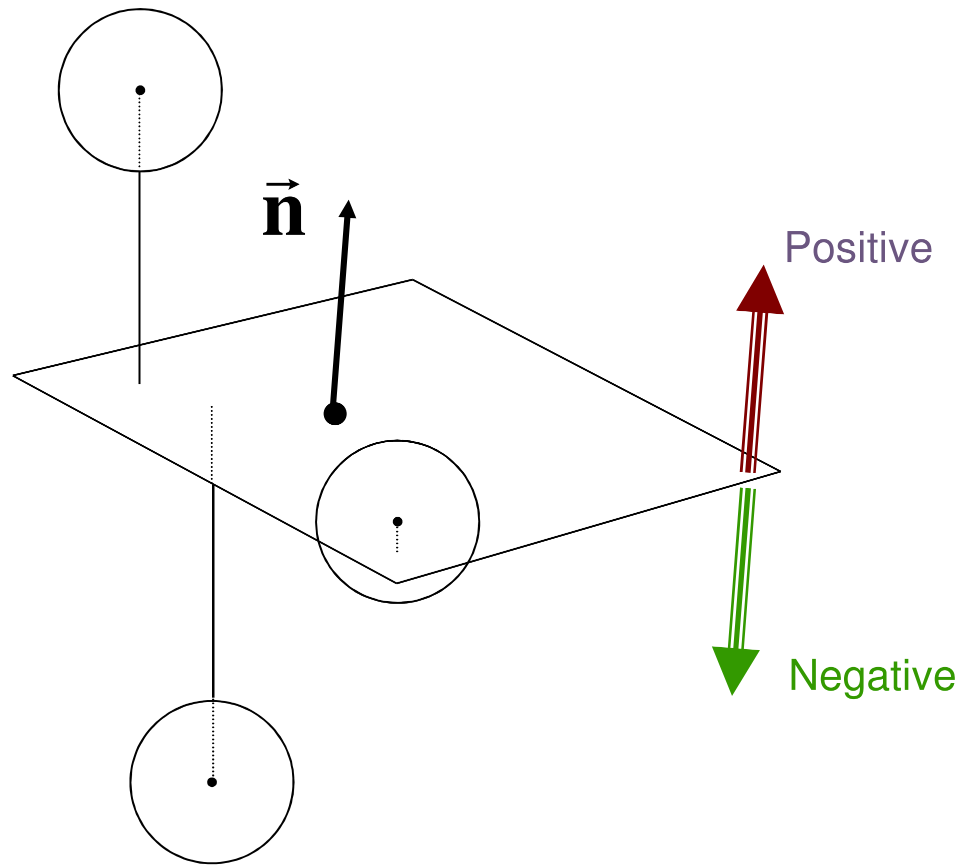
- Normal of each plane points outside
  - "outside" means positive distance
  - "inside" means negative distance



UCSD

Test Sphere and Plane

- For sphere with radius r and origin $\mathbf{x}$, test the distance to the origin, and see if it is beyond the radius
- Three cases:
  - $dist(\mathbf{x})>r$
    - completely above
  - $dist(\mathbf{x})<-r$
    - completely below
  - $-r<dist(\mathbf{x})<r$
    - intersects

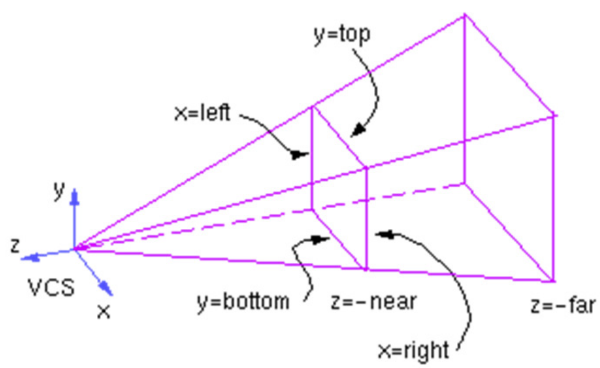


Culling Summary

- Precompute the normal $\mathbf{n}$ and value d for each of the six planes.
- Given a sphere with center $\mathbf{x}$ and radius r
- For each plane:
  - if $dist(\mathbf{x}) > r$: sphere is outside! (no need to continue loop)
  - add 1 to count if $dist(\mathbf{x})<-r$
- If we made it through the loop, check the count:
  - if the count is 6, the sphere is completely inside
  - otherwise the sphere intersects the frustum
  - *(can use a flag instead of a count)*

Culling Groups of Objects

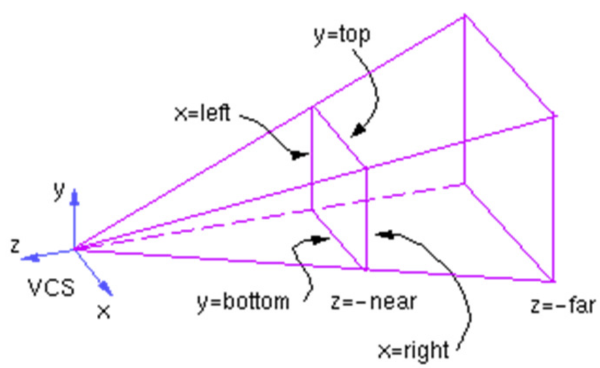
- Want to be able to cull the whole group quickly
- But if the group is partly in and partly out, want to be able to cull individual objects

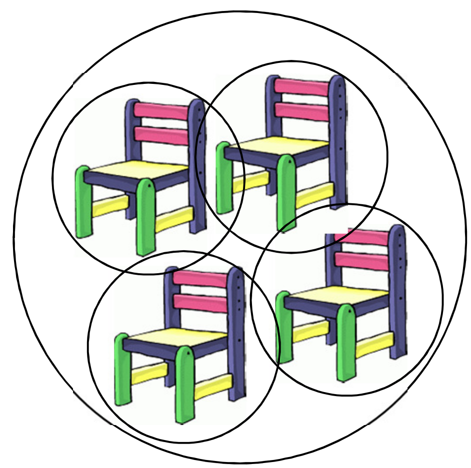


UCSD

Hierarchical Bounding Volumes

- Given hierarchy of objects
- Bounding volume of each node encloses the bounding volumes of all its children
- Start by testing the outermost bounding volume
  - If it is entirely outside, don't draw the group at all
  - If it is entirely inside, draw the whole group

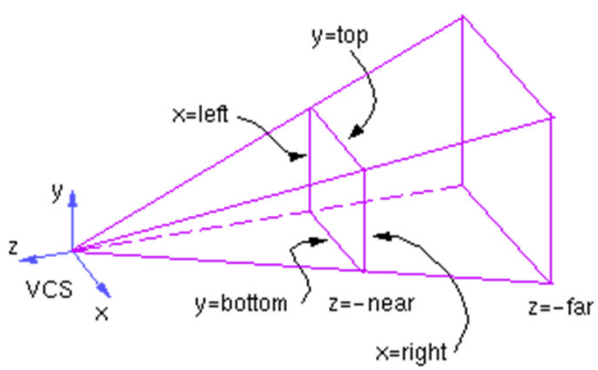


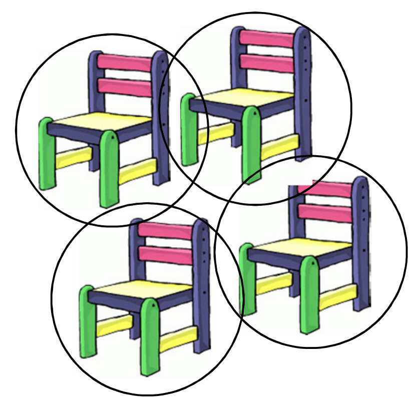


UCSD

Hierarchical Culling

- If the bounding volume is partly inside and partly outside
  - Test each child's bounding volume individually
  - If the child is in, draw it; if it's out cull it; if it's partly in and partly out, recurse.
  - If recursion reaches a leaf node, draw it normally

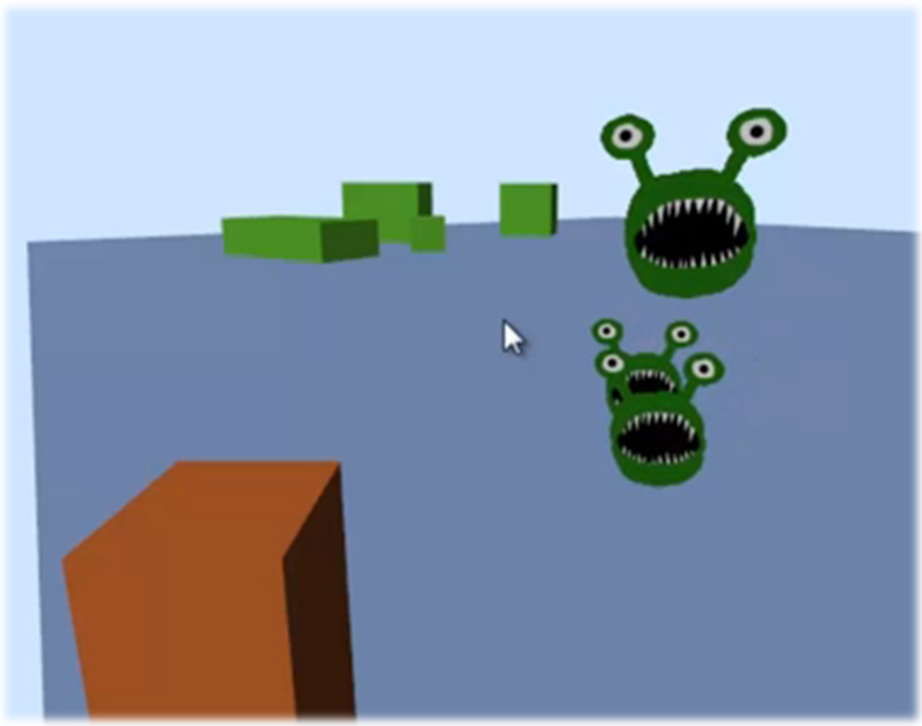




UCSD

Video

- Math for Game Developers - Frustum Culling
  - http://www.youtube.com/watch?v=4p-E_31XOPM



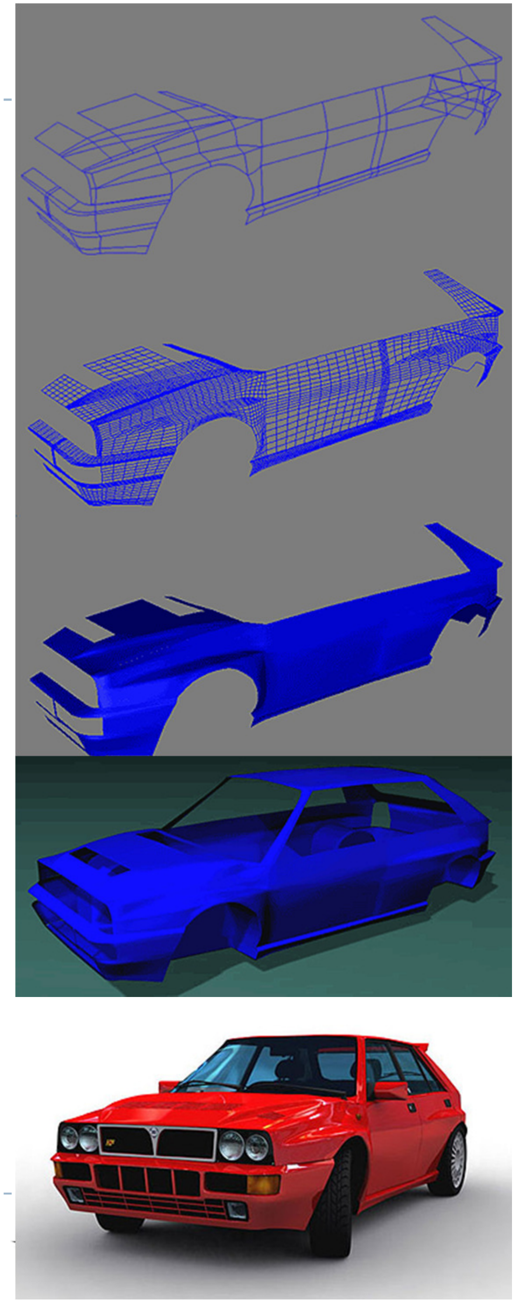
Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions
- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves

Modeling

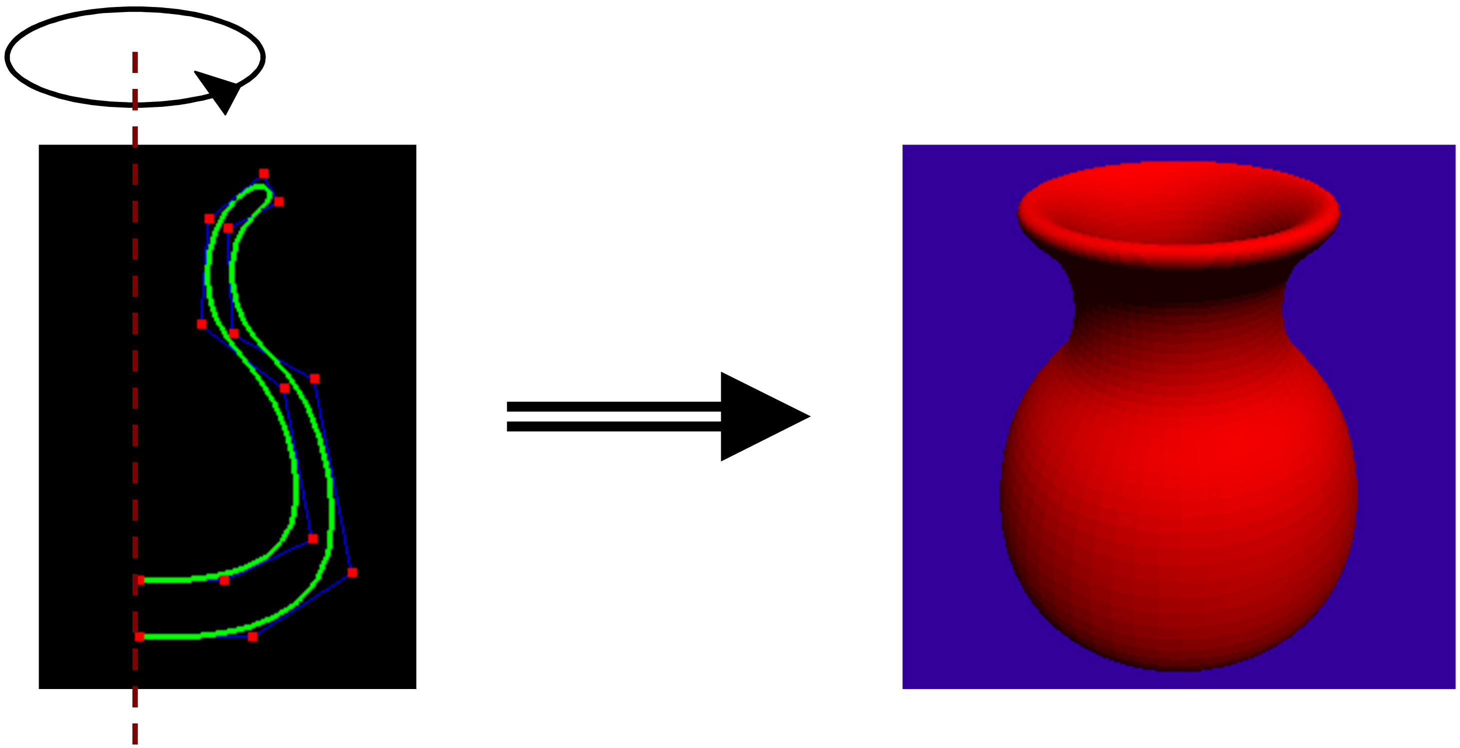
- Creating 3D objects
- How to construct complex surfaces?
- Goal
  - Specify objects with control points
  - Objects should be visually pleasing (smooth)
- Start with curves, then generalize to surfaces

- Next: What can curves be used for?



Curves

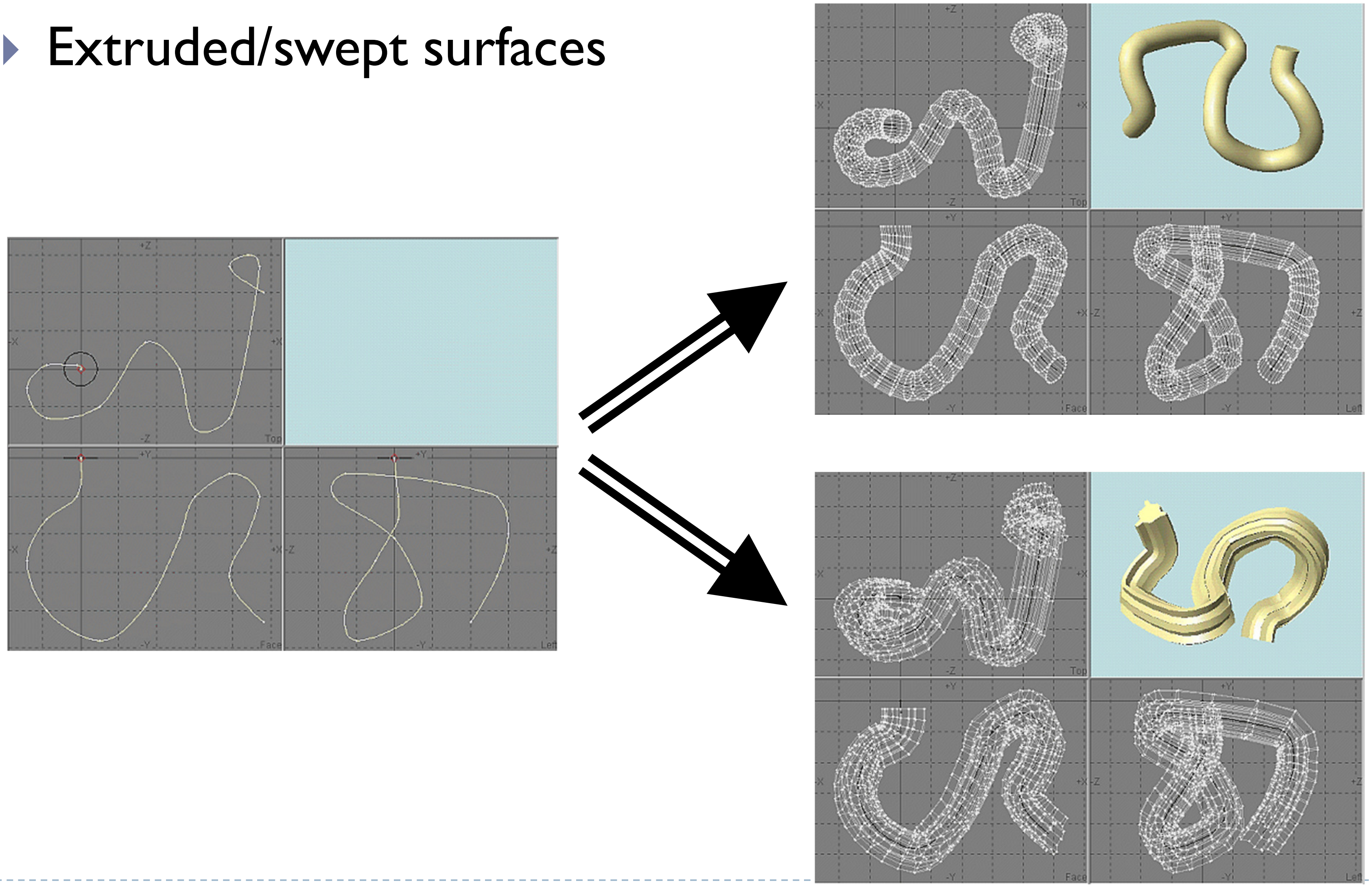
- Surface of revolution



UCSD

Curves

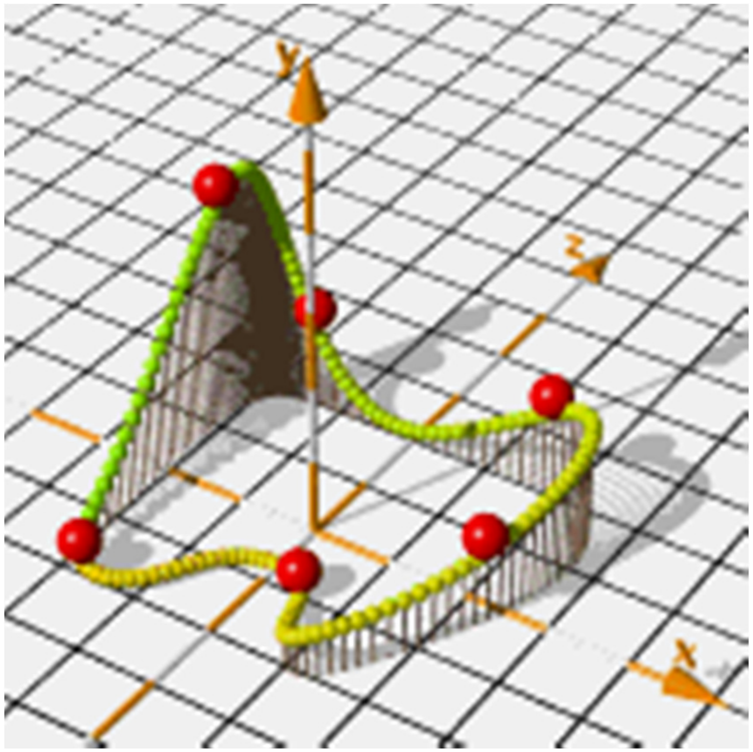
- Extruded/swept surfaces



UCSD

Curves

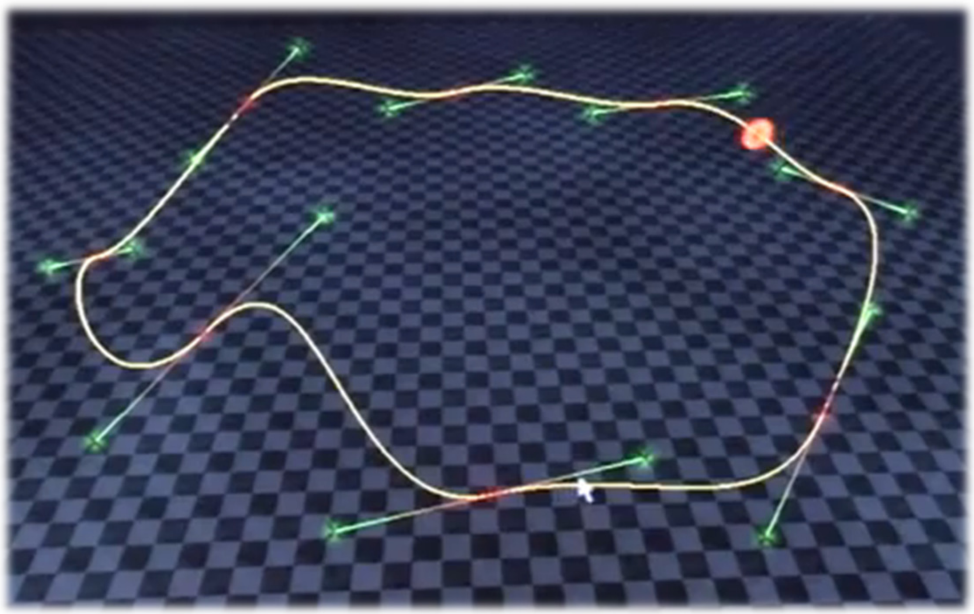
- Animation
  - Provide a "track" for objects
  - Use as camera path



UCSD

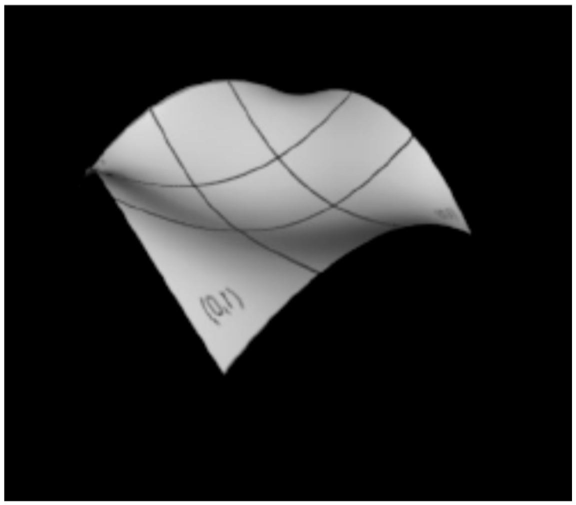
Video

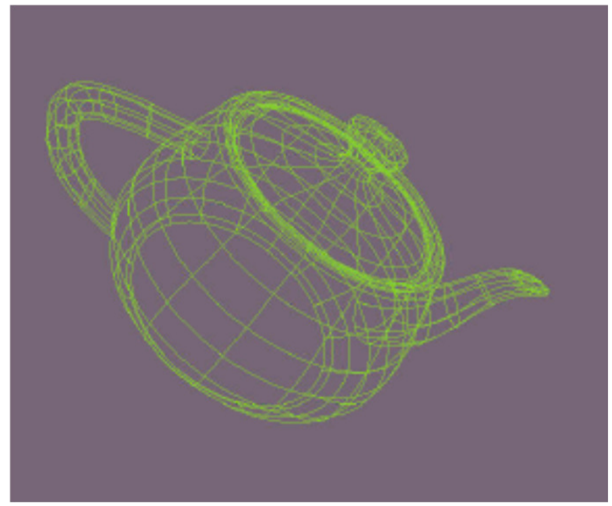
- Bezier Curves
  - http://www.youtube.com/watch?v=hlDYJNEiYvU



Curves

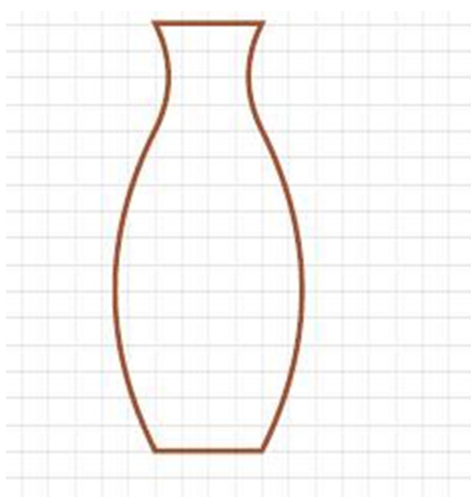
- Can be generalized to surface patches

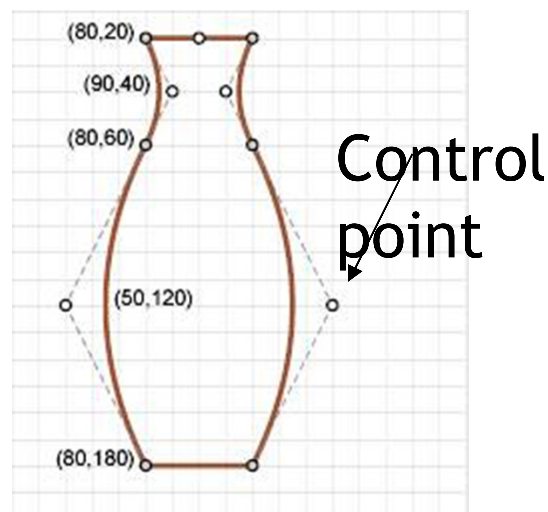




Curve Representation

- Specify many points along a curve, connect with lines?
  - Difficult to get precise, smooth results across magnification levels
  - Large storage and CPU requirements
  - How many points are enough?
- Specify a curve using a small number of "control points"
  - Known as a *spline curve* or just *spline*

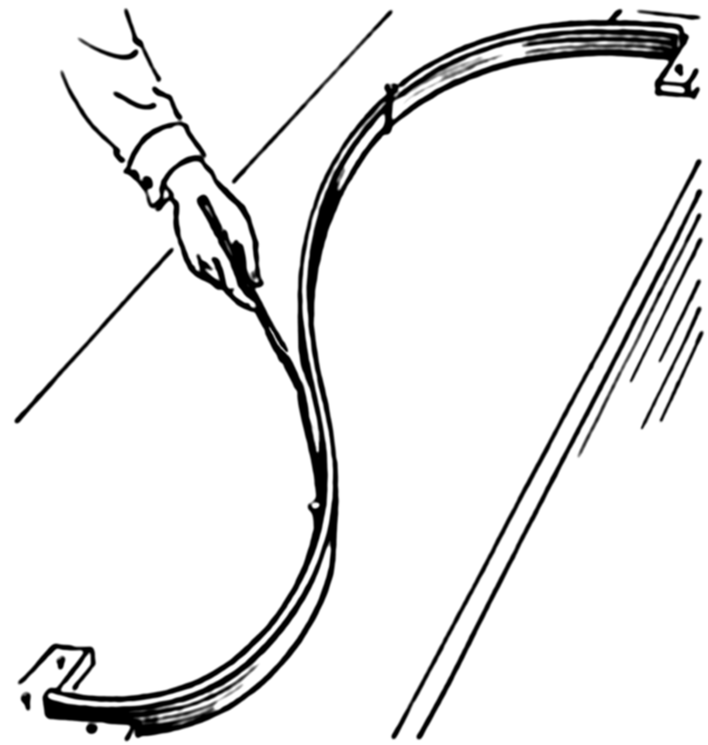




UCSD

Spline: Definition

- Wikipedia:
  - Term comes from flexible spline devices used by shipbuilders and draftsmen to draw smooth shapes.
  - Spline consists of a long strip fixed in position at a number of points that relaxes to form a smooth curve passing through those points.



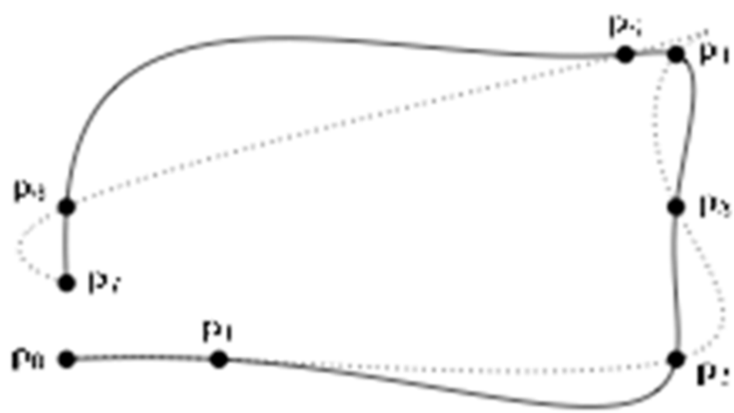
UCSD

Lecture Overview

- Polynomial Curves
  - Introduction
  - Polynomial functions
- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves

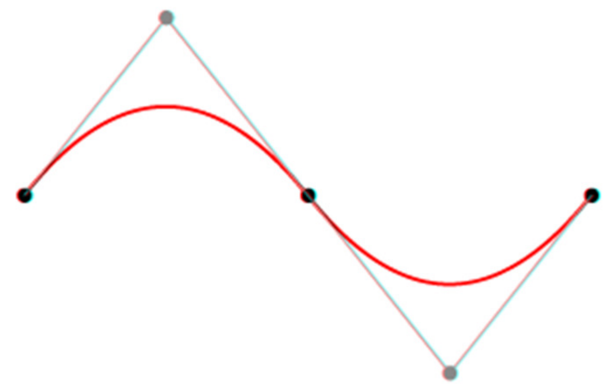
Interpolating Control Points

- "Interpolating" means that curve goes through all control points
- Seems most intuitive
- Surprisingly, not usually the best choice
  - Hard to predict behavior
  - Hard to get aesthetically pleasing curves



Approximating Control Points

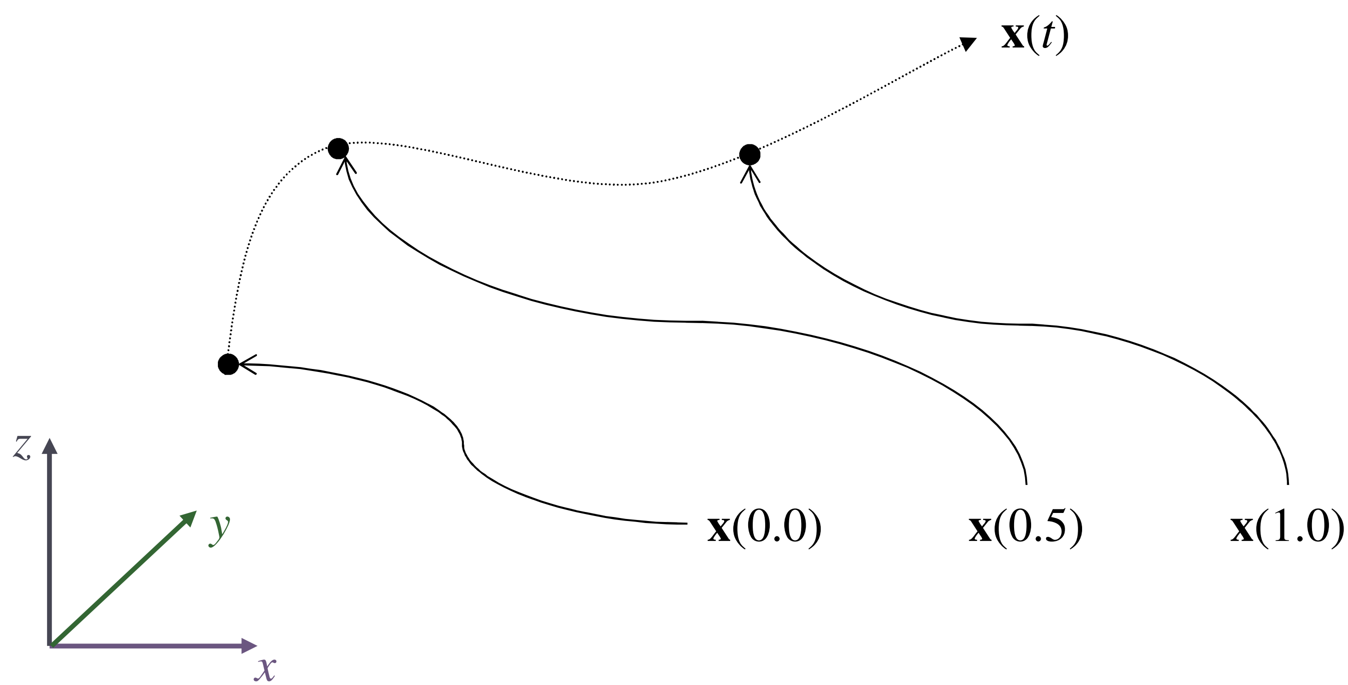
- Curve is "influenced" by control points



- Various types
- Most common: polynomial functions
  - Bézier spline (our focus)
  - B-spline (generalization of Bézier spline)
  - NURBS (Non Uniform Rational Basis Spline): used in CAD tools

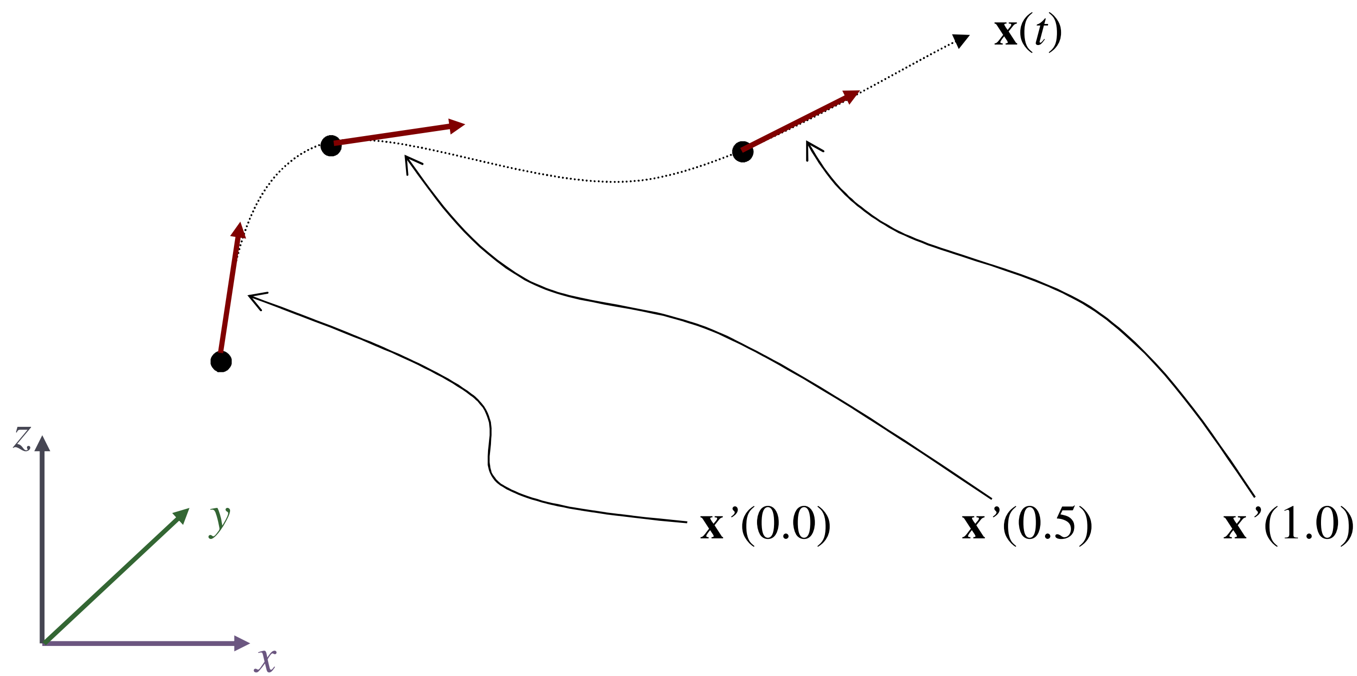
Mathematical Definition

- A vector valued function of one variable $\mathbf{x}(t)$
  - Given t, compute a 3D point $\mathbf{x}=(x,y,z)$
  - Could be interpreted as three functions: $x(t)$, $y(t)$, $z(t)$
  - Parameter t "moves a point along the curve"



Tangent Vector

- Derivative $\mathbf{x}'(t) = \frac{d\mathbf{x}}{dt} = (x'(t), y'(t), z'(t))$
- Vector x' points in direction of movement
- Length corresponds to speed

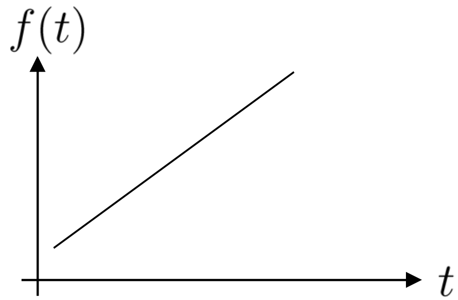


Lecture Overview

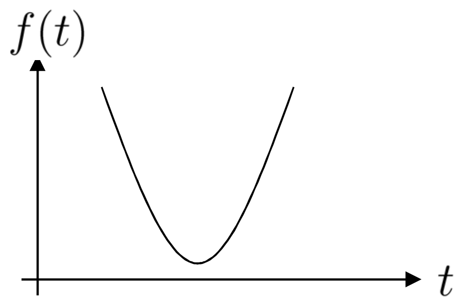
- Polynomial Curves
  - Introduction
  - Polynomial functions
- Bézier Curves
  - Introduction
  - Drawing Bézier curves
  - Piecewise Bézier curves

Polynomial Functions

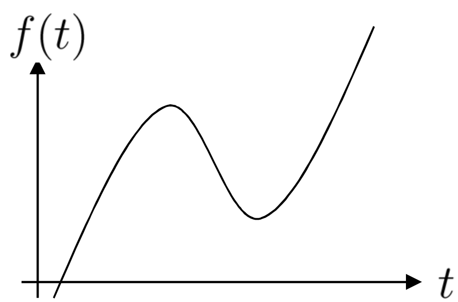
- Linear: $f(t) = at + b$
  (1st order)



- Quadratic: $f(t) = at^2 + bt + c$
  (2nd order)



- Cubic: $f(t) = at^3 + bt^2 + ct + d$
  (3rd order)



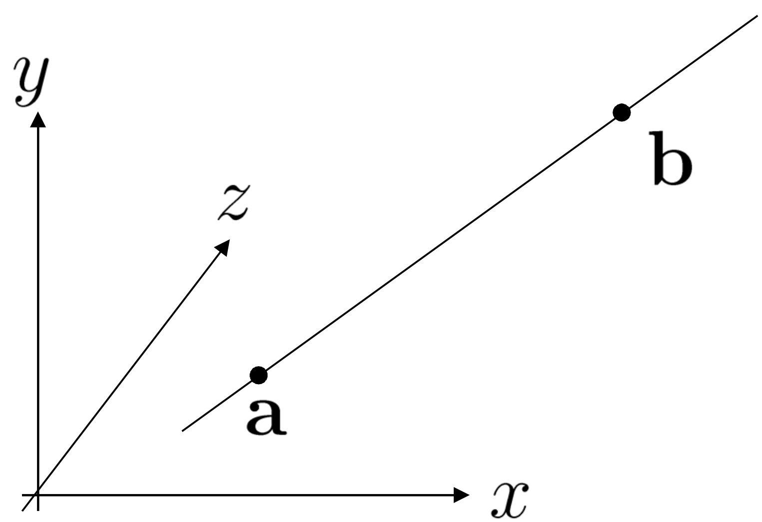
Polynomial Curves

- Linear $\mathbf{x}(t) = \mathbf{a}t + \mathbf{b}$

$$\mathbf{x} = (x, y, z), \mathbf{a} = (a_x, a_y, a_z), \mathbf{b} = (b_x, b_y, b_z)$$

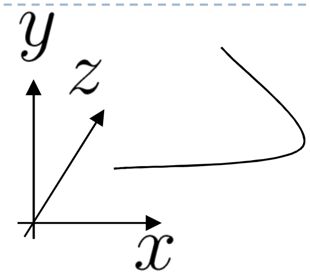
- Evaluated as:

$$x(t) = a_x t + b_x$$
$$y(t) = a_y t + b_y$$
$$z(t) = a_z t + b_z$$

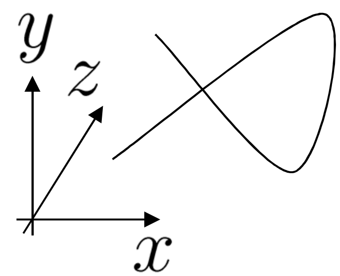


Polynomial Curves

- Quadratic: $\mathbf{x}(t) = \mathbf{a}t^2 + \mathbf{b}t + \mathbf{c}$
  (2nd order)



- Cubic: $\mathbf{x}(t) = \mathbf{a}t^3 + \mathbf{b}t^2 + \mathbf{c}t + \mathbf{d}$
  (3rd order)



- We usually define the curve for $0 \leq t \leq 1$

Control Points

- Polynomial coefficients $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ can be interpreted as *control points*
  - Remember: $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$ have x,y,z components each
- Unfortunately, they do not intuitively describe the shape of the curve
- Goal: intuitive control points

Control Points

- How many control points?
  - Two points define a line (1st order)
  - Three points define a quadratic curve (2nd order)
  - Four points define a cubic curve (3rd order)
  - $k+1$ points define a k-order curve
- Let's start with a line…

First Order Curve

- Based on linear interpolation (LERP)
  - Weighted average between two values
  - "Value" could be a number, vector, color, …
- Interpolate between points $\mathbf{p_0}$ and $\mathbf{p_1}$ with parameter t
  - Defines a "curve" that is straight (first-order spline)
  - $t=0$ corresponds to $\mathbf{p_0}$
  - $t=1$ corresponds to $\mathbf{p_1}$
  - $t=0.5$ corresponds to midpoint

$$\mathbf{x}(t) = Lerp\left(t,\ \mathbf{p}_0,\ \mathbf{p}_1\right) = \left(1-t\right)\mathbf{p}_0 + t\ \mathbf{p}_1$$

Linear Interpolation

- Three equivalent ways to write it
  - Expose different properties

1. Regroup for points **p**

$$\mathbf{x}(t) = \mathbf{p}_0(1-t) + \mathbf{p}_1 t$$

2. Regroup for t

$$\mathbf{x}(t) = (\mathbf{p}_1 - \mathbf{p}_0)t + \mathbf{p}_0$$

3. Matrix form

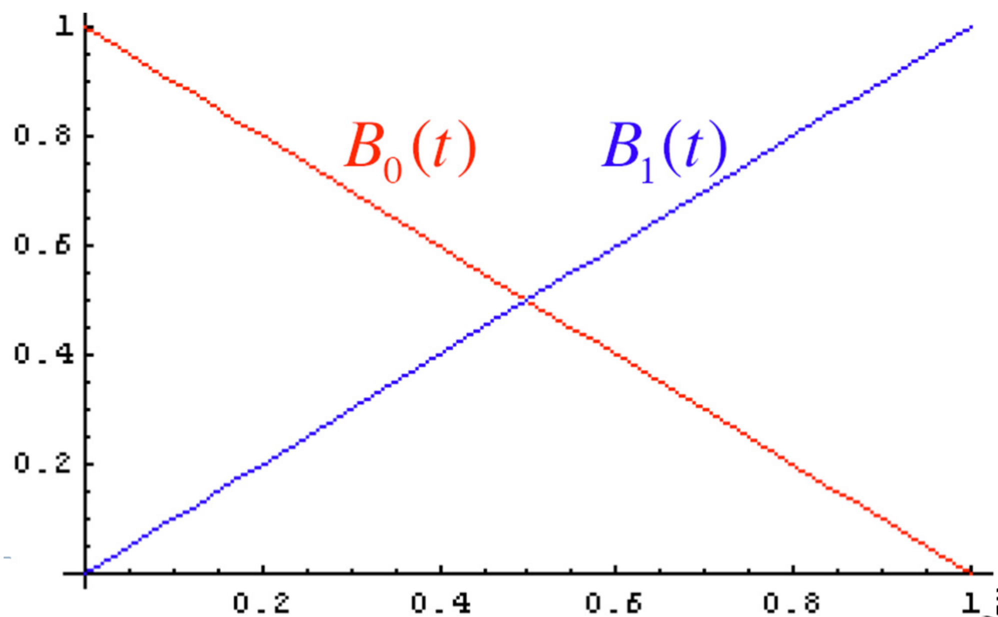
$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix}$$

Weighted Average

$$\mathbf{x}(t) = (1-t)\mathbf{p}_0 + \quad (t)\mathbf{p}_1$$

$$= B_0(t)\ \mathbf{p}_0 + B_1(t)\mathbf{p}_1, \text{ where } B_0(t) = 1-t \text{ and } B_1(t) = t$$

- Weights are a function of t
  - Sum is always 1, for any value of t
  - Also known as *blending functions*

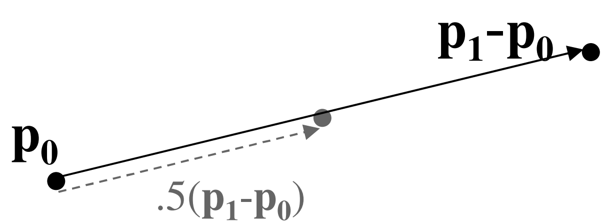


UCSD

Linear Polynomial

$$\mathbf{x}(t) = \underbrace{(\mathbf{p}_1 - \mathbf{p}_0)}_{\substack{\text{vector} \\ \mathbf{a}}} \, t + \underbrace{\mathbf{p}_0}_{\substack{\text{point} \\ \mathbf{b}}}$$

- Curve is based at point $\mathbf{p_0}$
- Add the vector, scaled by t



Matrix Form

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix} = \mathbf{GBT}$$

- Geometry matrix $\mathbf{G} = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 \end{bmatrix}$
- Geometric basis $\mathbf{B} = \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$
- Polynomial basis $T = \begin{bmatrix} t \\ 1 \end{bmatrix}$
- In components

$$\mathbf{x}(t) = \begin{bmatrix} p_{0x} & p_{1x} \\ p_{0y} & p_{1y} \\ p_{0z} & p_{1z} \end{bmatrix} \begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} t \\ 1 \end{bmatrix}$$