3D OBJECT REPRESENTATIONS

CEng 477

Introduction to Computer Graphics

Fall 2007-2008

Object Representations

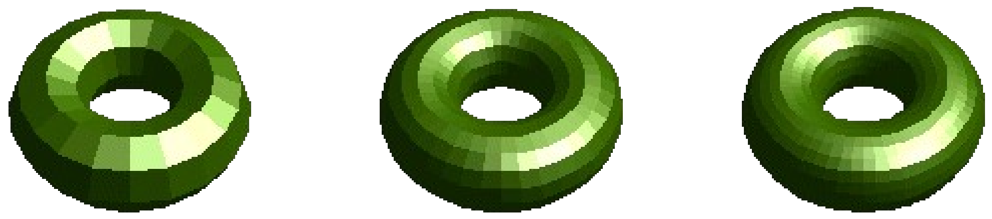
- Types of objects:
  geometrical shapes, trees, terrains, clouds, rocks, glass, hair, furniture, human body, etc.
- Not possible to have a single representation for all
  - Polygon surfaces
  - Spline surfaces
  - Procedural methods
  - Physical models
  - Solid object models
  - Fractals
  - ……

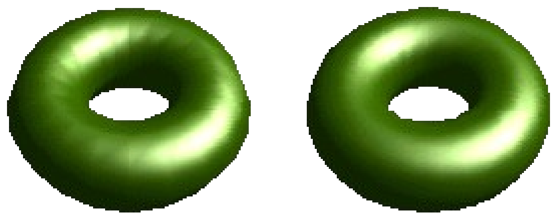
Two categories

- 3D solid object representations can be generally classified into two broad categories
  - Boundary representations
    - Inside and outside of objects are defined by this representation. E.g., polygon facets, spline patches
  - Space-partitioning representations
    - The inside of the object is divided into non-overlapping regions and the object is represented as a collection of these interior components. E.g., octree representation

Polygon Surfaces (Polyhedra)

- Set of adjacent polygons representing the object exteriors.
- All operations linear, so fast.
- Non-polyhedron shapes can be approximated by polygon meshes.
- Smoothness is provided either by increasing the number of polygons or interpolated shading methods.


Levels of detail


Interpolated shading

Data Structures

- Data structures for representing polygon surfaces:
  - Efficiency
    - Intersection calculations
    - Normal calculations
    - Access to adjacent polygons
  - Flexibility
    - Interactive systems
    - Adding, changing, removing vertices, polygons
  - Integrity

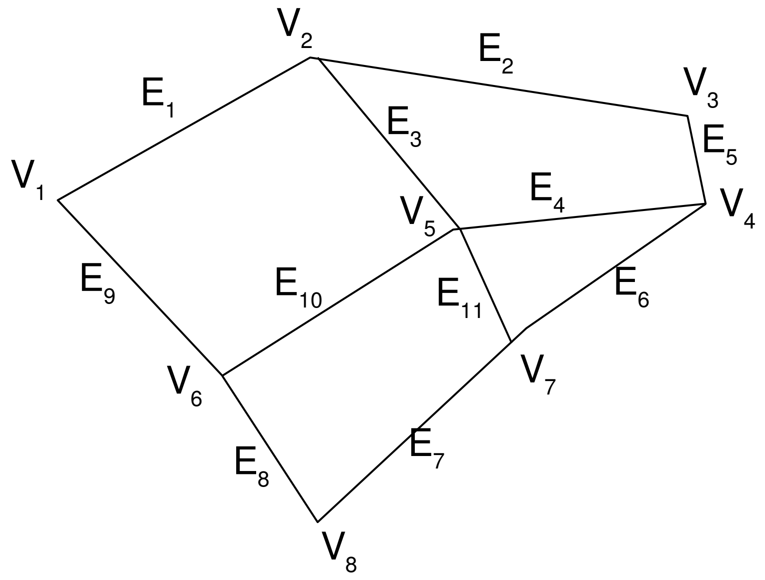
Polygon Tables

- Vertices Edges Polygons

V_1:(x_1,y_1,z_1)
V_2:(x_2,y_2,z_2)
V_3:(x_3,y_3,z_3)
V_4:(x_4,y_4,z_4)
V_5:(x_5,y_5,z_5)
V_6:(x_6,y_6,z_6)
V_7:(x_7,y_7,z_7)
V_8:(x_8,y_8,z_8)

E_1: V_1,V_2
E_2: V_2,V_3
E_3: V_2,V_5
E_4: V_4,V_5
E_5: V_3,V_4
E_6: V_4,V_7
E_7: V_7,V_8
E_8: V_6,V_8
E_9: V_1,V_6
E_{10}: V_5,V_6
E_{11}: V_5,V_7

S_1: E_1,E_3,E_{10},E_9
S_2: E_2,E_5,E_4,E_3
S_3: E_{10},E_{11},E_7,E_8
S_4: E_4,E_6,E_{11}



- Forward pointers: i.e. to access adjacent surfaces edges

V_1: E_1,E_9
V_2: E_1,E_2,E_3
V_3: E_2,E_5
V_4: E_4,E_5,E_6
V_5: E_3,$E_4$$E_{10}$,$E_{11}$
V_6: E_8,E_9,E_{10}
V_7: E_6,E_7,E_{11}
V_8: E_7,E_8

E_1: S_1	
E_2: S_2	
E_3: S_1,S_2	
E_4: S_2,S_4	
E_5: S_2	E_6: S_4
E_7: S_3	E_8: S_3
E_9: S_1	E_{10}: S_1,S_3
E : S,S	

- Additional geometric properties:
  - Slope of edges
  - Normals
  - Extends (bounding box)
- Integrity checks

$$\forall V, \ \exists E_a, E_b \, such \ that \ V \in E_a, V \in E_b$$

$$\forall E, \ \exists S \ such \ that \ E \in S$$

$$\forall S, \ \ S \ is \ closed$$

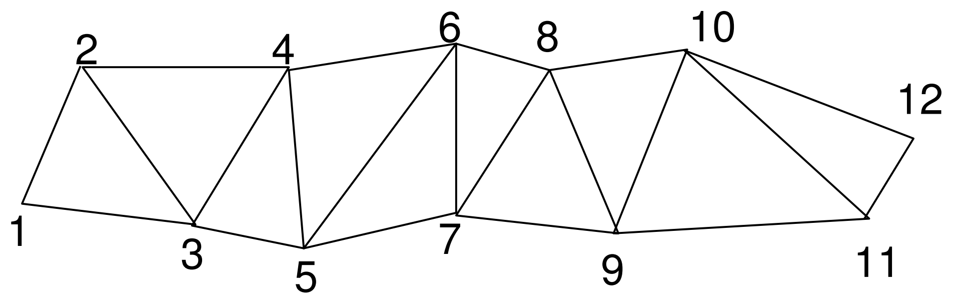
$$\forall S_{1,} \ \exists S_2 \ such \ that \ S_1 \cap S_2 \neq \emptyset$$

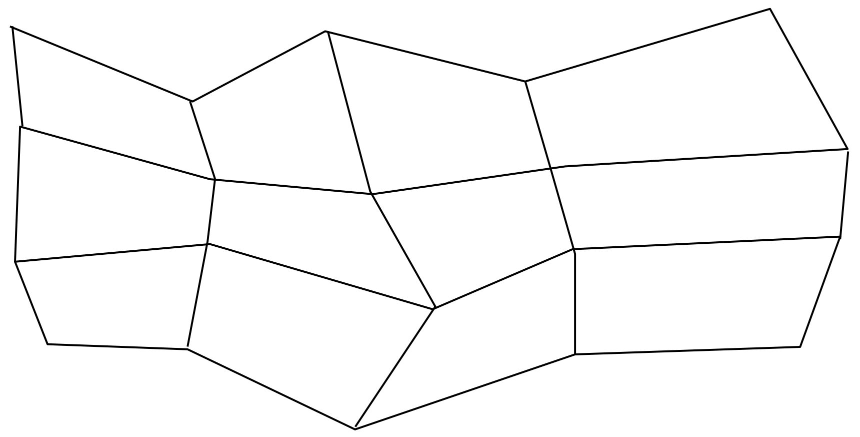
$$S_k \ is \ listed \ in \ E_m \Leftrightarrow E_m \ is \ listed \ in \ S_k$$

Polygon Meshes

- Triangle strips:
  123, 234, 345, ..., 10 11 12

  1 2 3 4 5 6 7 8 9 10 11 12
- Quadrilateral meshes:
  n×m array of vertices





Plane Equations

- Equation of a polygon surface:

$$A\,x+B\,y+C\,z+D=0$$

Linear set of equations:

$$(A/D)\,x_k+(B/D)\,y_k+(C/D)\,z_k=-1, \qquad k=1,2,3$$

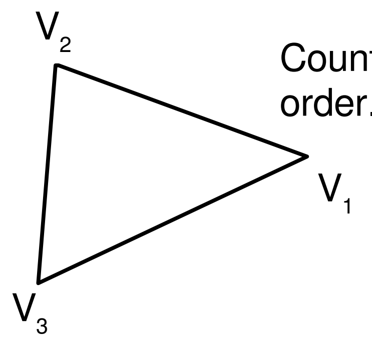
$$A=y_1(z_2-z_3)+y_2(z_3-z_1)+y_3(z_1-z_2)$$
$$B=z_1(x_2-x_3)+z_2(x_3-x_1)+z_3(x_1-x_2)$$
$$C=x_1(y_2-y_3)+x_2(y_3-y_1)+x_3(y_1-y_2)$$
$$D=-x_1(y_2 z_3-y_3 z_2)-x_2(y_3 z_1-y_1 z_3)-x_3(y_1 z_2-y_2 z_1)$$

- Surface Normal:

$$\boldsymbol{N}=(A,B,C)$$

extracting normal from vertices:

$$\boldsymbol{N}=(V_2-V_1)\times(V_3-V_1)$$



Counterclockwise order.

- Find plane equation from normal and a point on the surface

$$\begin{aligned}
&(A, B, C) = \boldsymbol{N} \\
&\boldsymbol{N} \cdot (x, y, z) + D = 0 \\
&\boldsymbol{P} \ \textit{is a point in the surface (i.e. a vertex)} \\
&D = -\boldsymbol{N} \cdot \boldsymbol{P}
\end{aligned}$$

- Inside outside tests of the surface (N is pointing towards outside):

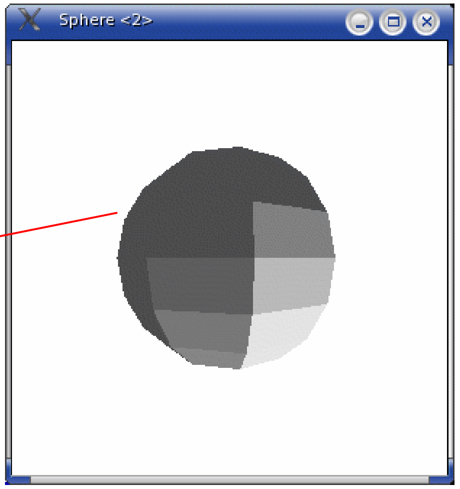
$$\begin{aligned}
&A x + B y + C z + D < 0, \quad \textit{point is inside the surface} \\
&A x + B y + C z + D > 0, \quad \textit{point is outside the surface}
\end{aligned}$$

OpenGL Polyhedron Functions

- There are two methods in OpenGL for specifying polygon surfaces.
  - You can use geometric primitives, GL_TRIANGLES, GL_QUADS, etc. to describe the set of polygons making up the surface
  - Or, you can use the GLUT functions to generate five regular polyhedra in wireframe or solid form.

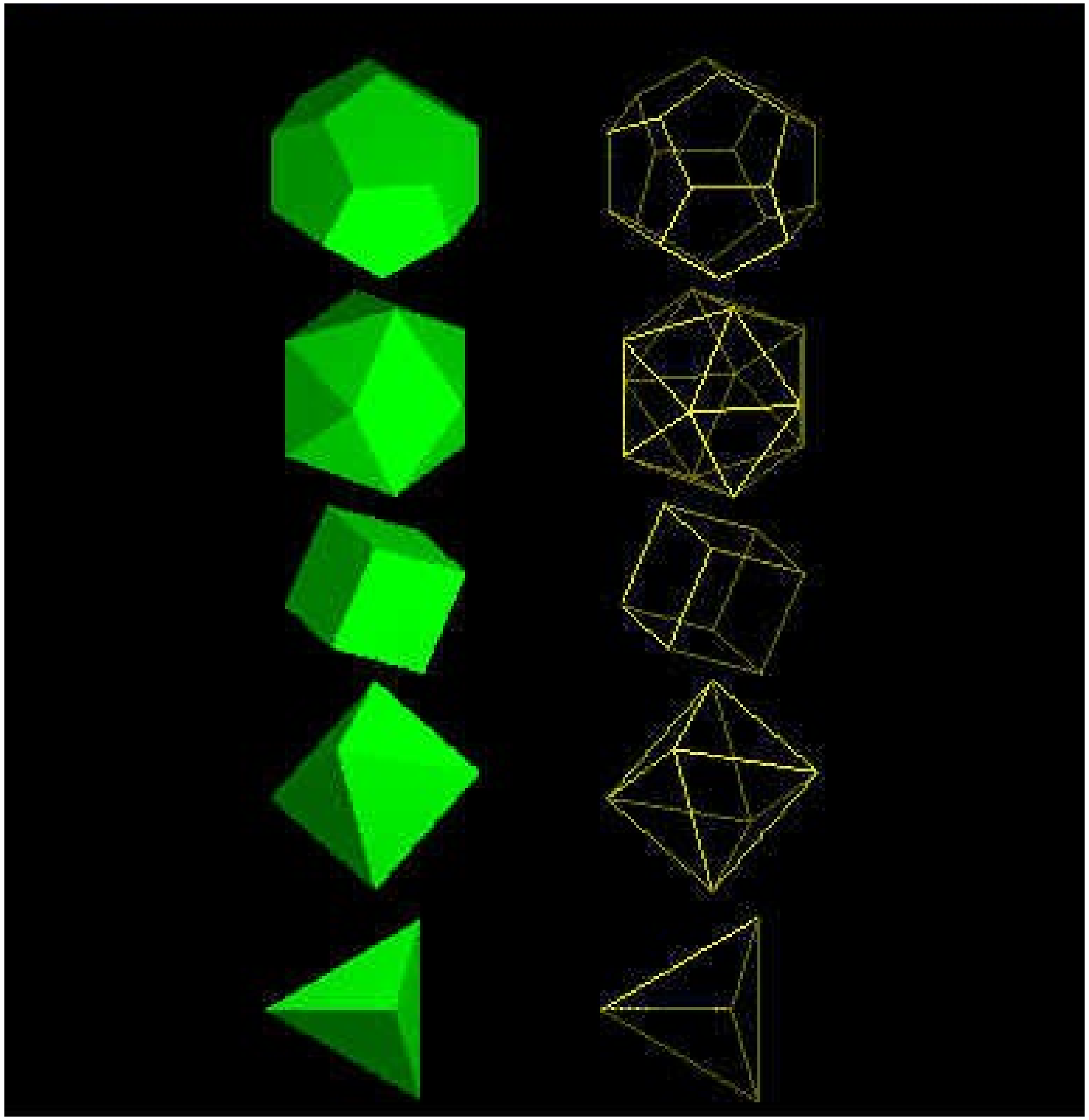
Drawing a sphere with GL_QUAD_STRIP

```
void drawSphere(double r, int lats, int longs) {
    int i, j;
    for(i = 0; i <= lats; i++) {
        double lat0 = M_PI * (-0.5 + (double) (i - 1) / lats);
        double z0 = sin(lat0);
        double zr0 = cos(lat0);
        double lat1 = M_PI * (-0.5 + (double) i / lats);
        double z1 = sin(lat1);
        double zr1 = cos(lat1);
        glBegin(GL_QUAD_STRIP);
        for(j = 0; j <= longs; j++) {
            double lng = 2 * M_PI * (double) (j - 1) / longs;
            double x = cos(lng);
            double y = sin(lng);
            glVertex3f(x * zr0, y * zr0, z0);
            glVertex3f(x * zr1, y * zr1, z1);
        }
        glEnd();
    }
}
```



You will not see it like this until you learn “lighting”.

Five regular polyhedra provided by GLUT



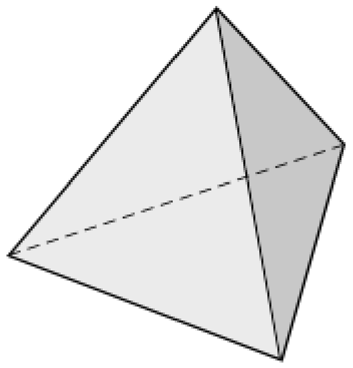
Also called Platonic solids.

The faces are identical regular polygons.

All edges, edge angles are equal.

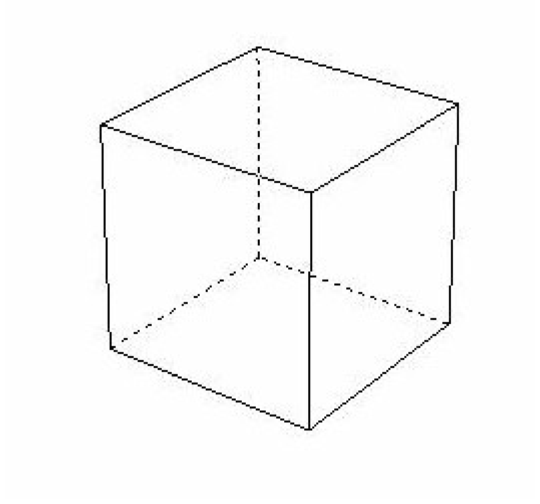
Tetrahedron

- glutWireTetrahedron ( );
- glutSolidTetrahedron ( );
- This polyhedron is generated with its center at the world-coordinate origin and with a radius equal to $\sqrt{3}$



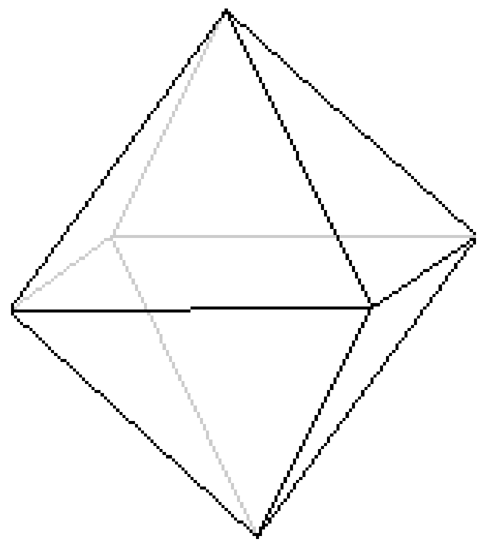
Cube

- glutWireCube (edgeLength);
- glutSolidCube (edgeLength);
- Creates a cube centered at the world-coordinate origin with the given edge length.



Octahedron

- glutWireOctahedron ( );
- glutSolidOctahedron ( );
- Creates a octahedron with 8 equilateral triangular faces. The radius is 1.

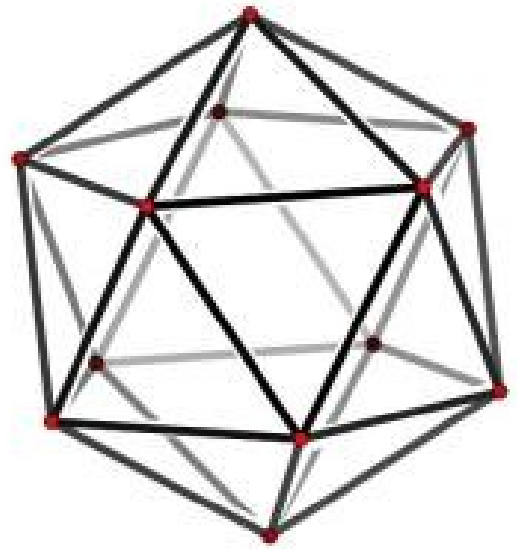


Dodecahedron

- glutWireDodecahedron ( );
- glutSolidDodecahedron ( );
- Creates a dodecahedron centered at the world-coordinate origin with 12 pentagon faces.

Icosahedron

- glutWireIcosahedron ( );
- glutSolidIcosahedron ( );
- Creates an icosahedron with 20 equilateral triangles. Center at origin and the radius is 1.



Curved Surfaces

- Can be represented by either parametric or non-parametric equations.
- Types of curved surfaces
  - Quadric surfaces
  - Superquadrics
  - Polynomial and Exponential Functions
  - Spline Surfaces

Quadric Surfaces

- Described with second degree (quadric) equations.
- Examples:
  - Spheres
  - Ellipsoids
  - Tori
  - Paraboloids
  - Hyperboloids
- Can also be created using spline representations.

Sphere

- Non-parametric equation

$$x^2 + y^2 + z^2 = r^2$$

- Parametric equation using latitude and longitude angles

$$x = r\cos\varphi\cos\theta, \qquad -\pi/2 \le \varphi \le \pi/2$$
$$y = r\cos\varphi\sin\theta, \qquad -\pi \le \theta \le \pi$$
$$z = r\sin\varphi$$

Ellipsoid

- Non-parametric equation

$$\left(\frac{x}{r_x}\right)^2+\left(\frac{y}{r_y}\right)^2+\left(\frac{z}{r_z}\right)^2=1$$

- Parametric equation using latitude and longitude angles

$$x=r_x\cos\varphi\cos\theta, \qquad -\pi/2\le\varphi\le\pi/2$$
$$y=r_y\cos\varphi\sin\theta, \qquad -\pi\le\theta\le\pi$$
$$z=r_z\sin\varphi$$

Superquadrics

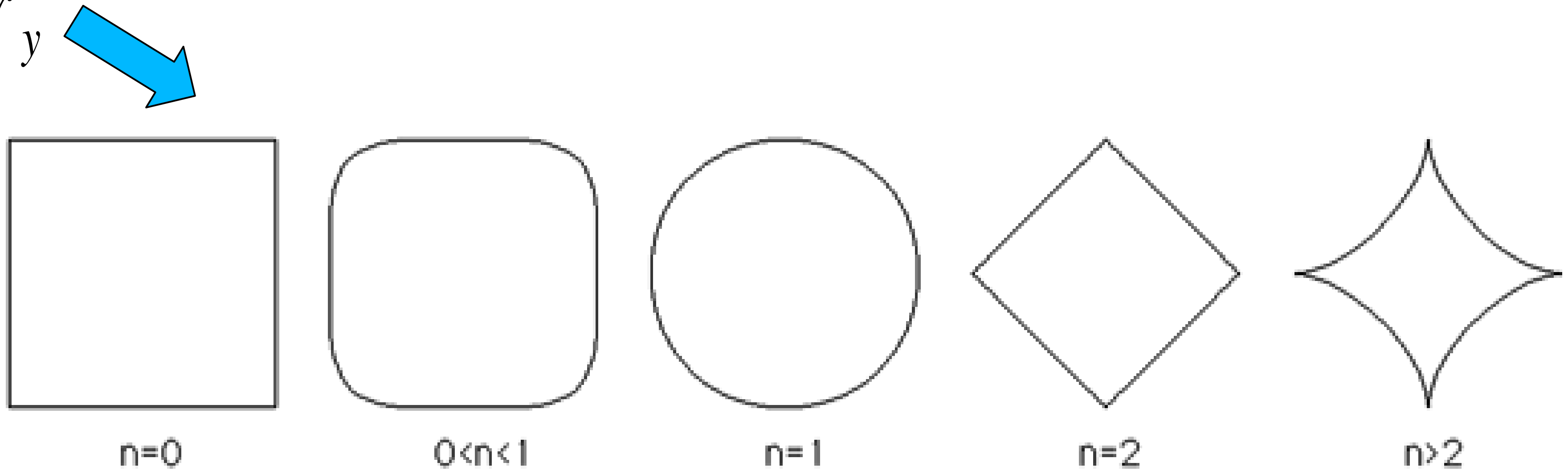
- Adding additional parameters to quadric representations to get new object shapes.
- One additional parameter is added to curve (i.e., 2d) equations and two parameters are added to surface (i.e., 3d) equations.

Superellipse

$$\left(\frac{x}{r_x}\right)^{2/n} + \left(\frac{y}{r_y}\right)^{2/n} = 1$$

$$x = r_x \cos^n \theta, \quad -\pi \le \theta \le \pi$$
$$y = r_y \sin^n \theta$$

$r_x = r_y$



Superellipse

- Used by industrial designers often

Superellipsoid

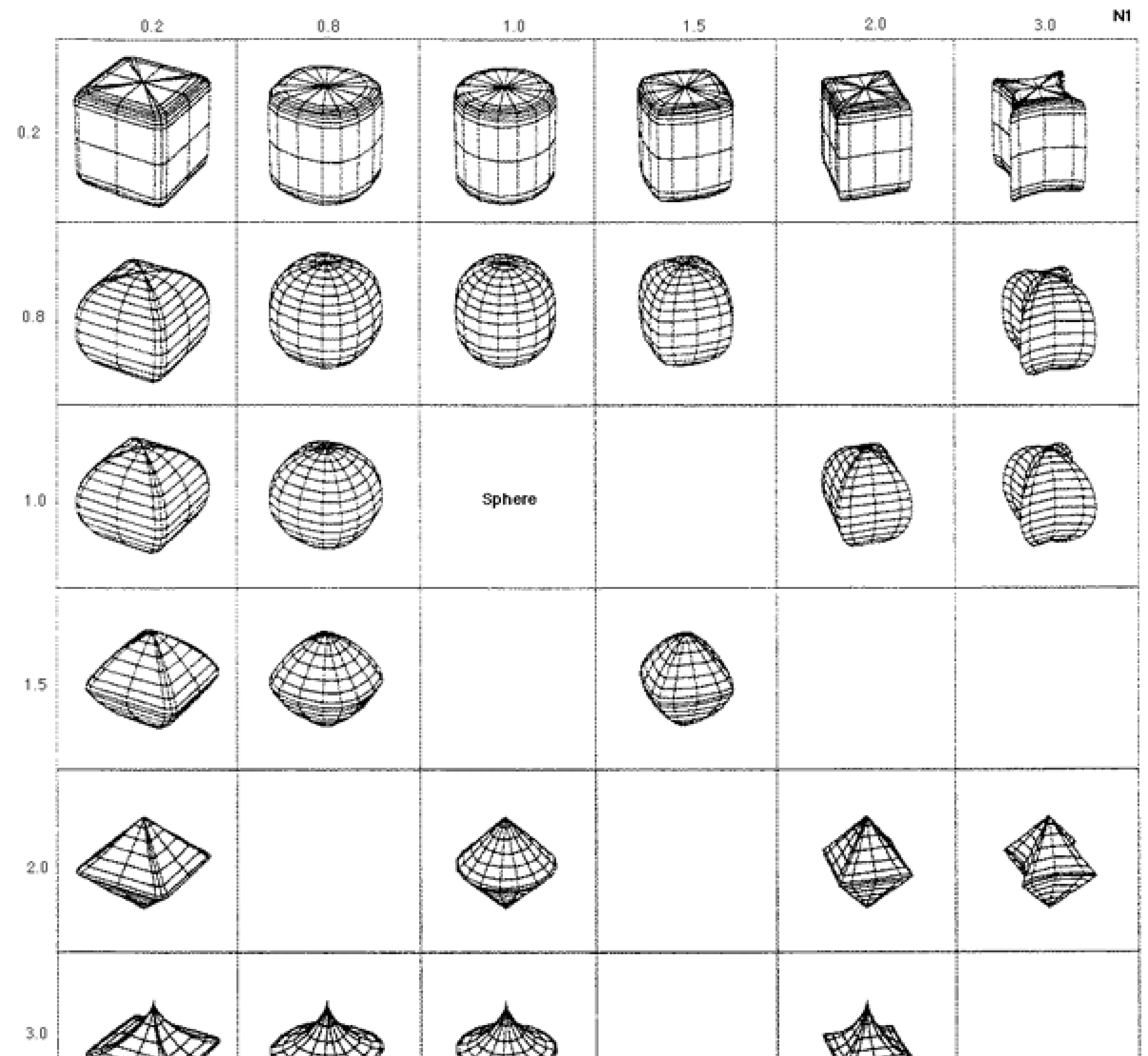
$$\left[\left(\frac{x}{r_x}\right)^{2/s_2}+\left(\frac{y}{r_y}\right)^{2/s_2}\right]^{s_2/s_1}+\left(\frac{z}{r_z}\right)^{2/s_1}=1$$

$$x=r_x\cos^{s_1}\varphi\cos^{s_2}\theta, \qquad -\pi/2\le\varphi\le\pi/2$$

$$y=r_y\cos^{s_1}\varphi\sin^{s_2}\theta, \qquad -\pi\le\theta\le\pi$$

$$z=r_z\sin^{s_1}\varphi$$

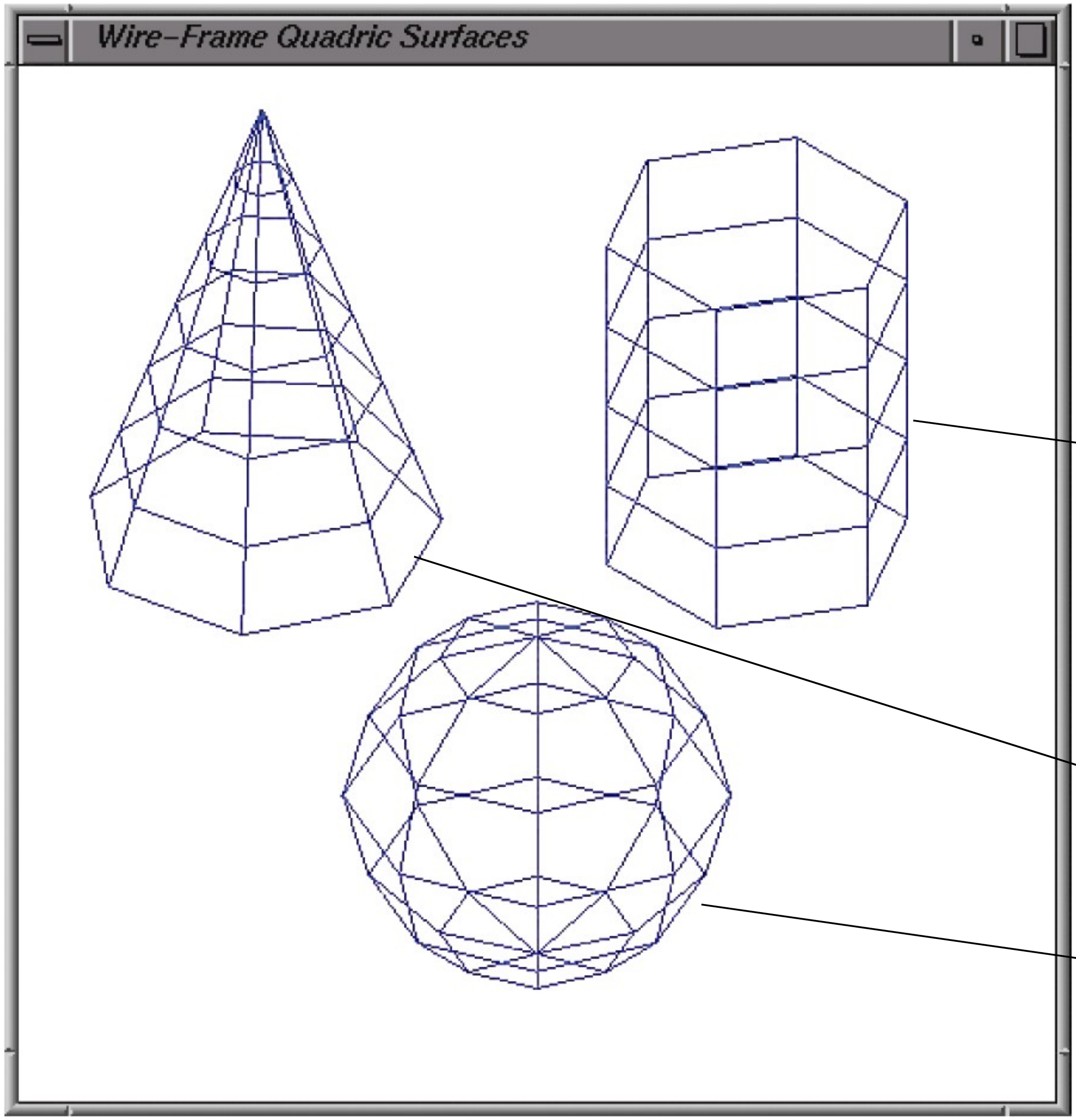
Superellipsoid



OpenGL Quadric-Surface and Cubic-Surface Functions

- GLUT and GLU provide functions to draw quadric-surface objects.
- GLUT functions
  - Sphere, cone, torus
- GLU functions
  - Sphere, cylinder, tapered cylinder, cone, flat circular ring (or hollow disk), and a section of a circular ring (or disk)
- GLUT also provides a function to draw a “teapot” (modeled with bicubic surface pathces).

Examples



GLU cylinder

GLUT cone

GLUT sphere

GLUT functions

- glutWireSphere (r, nLongitudes, nLatitudes);
- glutSolidSphere (r, nLongitudes, nLatitudes);
- glutWireCone(rBase, height, nLong, nLat);
- glutSolidCone(rBase, height, nLong, nLat);
- glutWireTorus(rCrossSection, rAxial, nConcentric, nRadial);
- glutWireTorus(rCrossSection, rAxial, nConcentric, nRadial);
- glutWireTeapot(size);
- glutSolidTeapot(size);

GLU Quadric-Surface Functions

- GLU functions are harder to use.
- You have to assign a name to the quadric.
- Activate the GLU quadric renderer
- Designate values for the surface parameters
- Example:

```
GLUquadricObj *mySphere;
mySphere = gluNewQuadric();
gluQuadricStyle (mySphere, GLU_LINE);
gluSphere (mySphere, r, nLong, nLat);
```

Quadric styles

- Other than GLU_LINE we have the following drawing styles:
  - GLU_POINT
  - GLU_SILHOUETTE
  - GLU_FILL

Other GLU quadric objects

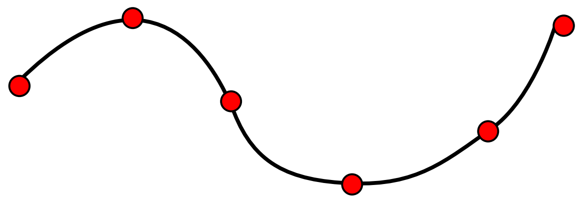
- gluCylinder (name, rBase, rTop, height, nLong, nLat);
- gluDisk (name, rInner, rOuter, nRadii, nRings);
- gluPartialDisk (… parameters …);

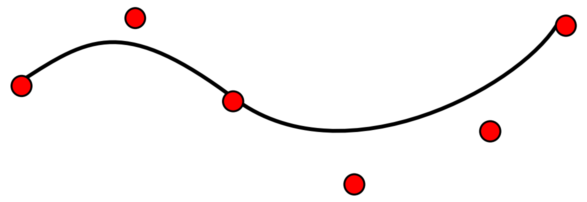
Additional functions to manipulate GLU quadric objects

- gluDeleteQuadric (name);
- gluQuadricOrientation (name, normalDirection);
  - To specify front/back directions.
  - normalVector is GLU_INSIDE or GLU_OUTSIDE
- gluQuadricNormals (name, generationMode);
  - Mode can be GLU_NONE, GLU_FLAT, or GLU_SMOOTH based on the lighting conditions you want to use for the quadric object.

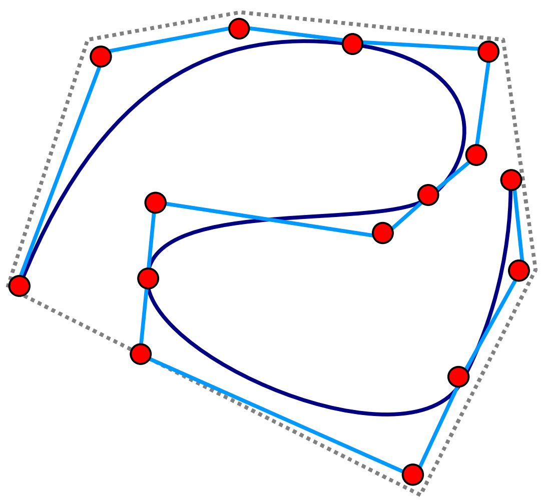
Spline Representations

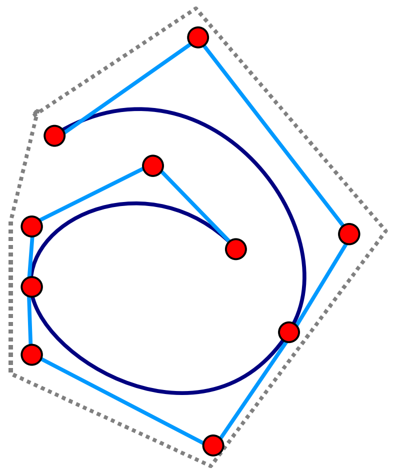
- Spline curve: Curve consisting of continous curve segments approximated or interpolated on polygon control points.
- Spline surface: a set of two spline curves matched on a smooth surface.
- Interpolated: curve passes through control points
- Approximated: guided by control points but not necessarily passes through them.


Interpolated


Approximated

- Convex hull of a spline curve: smallest polygon including all control points.
- Characteristic polygon, control path: vertices along the control points in the same order.

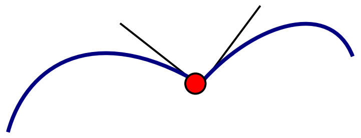




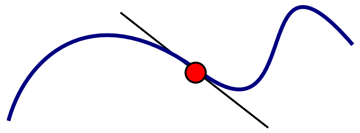
- Parametric equations:

$$x = x(u), \qquad y = y(u), \qquad z = z(u), \qquad u_1 \leq u \leq u_2$$

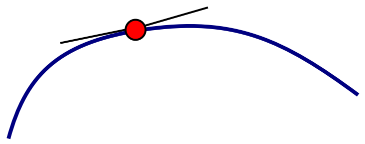
- Parametric continuity: Continuity properties of curve segments.
  - Zero order: Curves intersects at one end-point: C^0



  - First order: C^0 *and c*urves has same tangent at intersection: C^1



  - Second order: C^0 , C^1 *and c*urves has same second order derivative: C^2



- Geometric continuity:
  Similar to parametric continuity but only the direction of derivatives are significant. For example derivative (1,2) and (3,6) are considered equal.
- G^0, G^1, G^2 : zero order, first order, and second order geometric continuity.

Spline Equations

- Cubic curve equations:

$$x(u)=a_x u^3+b_x u^2+c_x u+d_x$$
$$y(u)=a_y u^3+b_y u^2+c_y u+d_y \qquad 0\le u\le 1$$
$$z(u)=a_z u^3+b_z u^2+c_z u+d_z$$

$$x(u)=\begin{bmatrix}u^3 & u^2 & u & 1\end{bmatrix}\cdot\begin{bmatrix}a_x\\ b_x\\ c_x\\ d_x\end{bmatrix}=U\cdot C$$

- General form: $x(u)=U\cdot M_s\cdot M_g$
- $\mathbf{M}_s$: spline transformation (blending functions) $\mathbf{M}_g$: geometric constraints (control points)

Natural Cubic Splines

- Interpolation of *n*+1 control points. *n* curve segments. 4*n* coefficients to determine
- Second order continuity. 4 equation for each of *n*-1 common points:

$$x_k(1)=p_k, \quad x_{k+1}(0)=p_k, \quad \{x'_k(1)=x'_{k+1}(0), \quad \{x ¿''_k(1)=x''_{k+1}(0)¿$$

  4*n* equations required, 4*n*-4 so far.

- Starting point condition, end point condition.

$$x_1(0)=p_0, \quad x_n(1)=p_n$$

- Assume second derivative 0 at end-points or add phantom control points p_{-1}, p_{n+1}.

$$x''_1(0)=0, \quad \{x''_n(1)=0¿$$

- Write 4*n* equations for 4*n* unknown coefficients and solve.
- Changes are not local. A control point effects all equations.
- Expensive. Solve 4*n* system of equations for changes.

Hermite Interpolation

- End point constraints for each segment is given as:

$$P(0)=p_k, \qquad P(1)=p_{k+1}, \qquad \{\boldsymbol{P}'(0)=\mathbf{Dp}_{\boldsymbol{k}}, \qquad \{\boldsymbol{P}\text{¿}'(1)=\mathbf{Dp}_{\boldsymbol{k+1}},\text{¿}$$

- Control point positions and first derivatives are given as constraints for each end-point. $P(0)=p_k$ $P(1)=p_{k+1}$ $P(0)=p_k,$ $P(1)=p_{k+1},$

$$P(u)=\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}\cdot\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \qquad P'(u)=\begin{bmatrix} 3\mathrm{u}^2 & 2\mathrm{u} & 1 & 0 \end{bmatrix}\cdot\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

$$\begin{bmatrix} p_k \\ p_{k+1} \\ \mathbf{Dp}_{\boldsymbol{k}} \\ \mathbf{Dp}_{\boldsymbol{k+1}} \end{bmatrix}=\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}\cdot\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} \qquad \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}=\begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1}\cdot\begin{bmatrix} p_k \\ p_{k+1} \\ \mathbf{Dp}_{\boldsymbol{k}} \\ \mathbf{Dp}_{\boldsymbol{k+1}} \end{bmatrix}$$

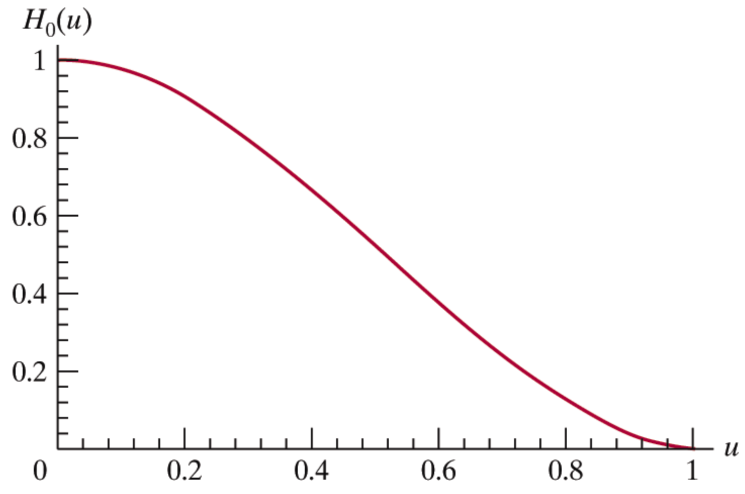
Hermite curves

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ \mathbf{Dp}_{\boldsymbol{k}} \\ \mathbf{Dp}_{\boldsymbol{k+1}} \end{bmatrix} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ \mathbf{Dp}_{\boldsymbol{k}} \\ \mathbf{Dp}_{\boldsymbol{k+1}} \end{bmatrix} = M_H \cdot \begin{bmatrix} p_k \\ p_{k+1} \\ \mathbf{Dp}_{\boldsymbol{k}} \\ \mathbf{Dp}_{\boldsymbol{k+1}} \end{bmatrix}$$

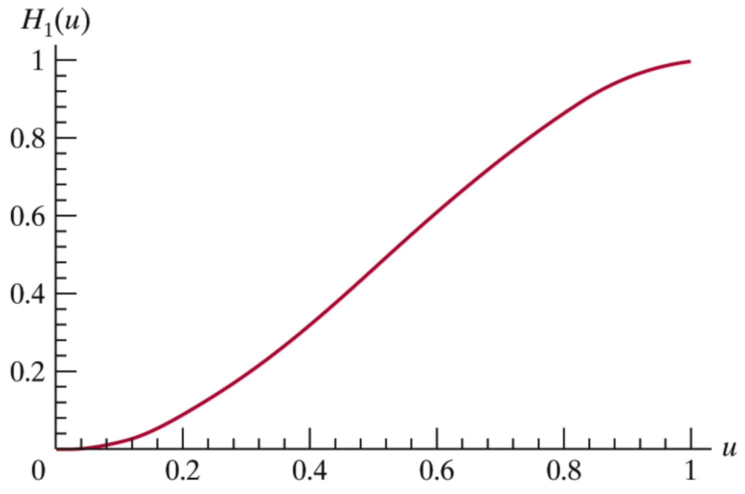
$$P(u) = p_k(2\mathrm{u}^3 - 3\mathrm{u}^2 + 1) + p_{k+1}(-2\mathrm{u}^3 + 3\mathrm{u}^2) + \mathbf{Dp}_{\boldsymbol{k}}(u^3 - 2\mathrm{u}^2 + u) + \mathbf{Dp}_{\boldsymbol{k+1}}(u^3 - u^2)$$

These polynomials are called Hermite blending functions, and tells us how to blend boundary conditions to generate the position of a point $\mathbf{P}(u)$ on the curve

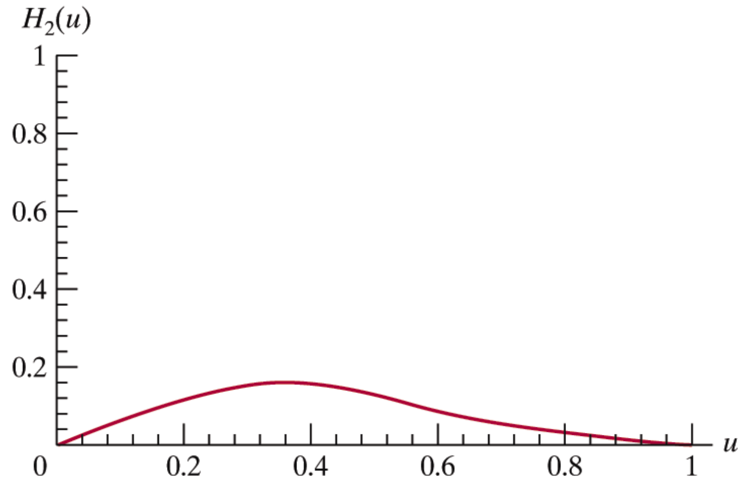
Hermite blending functions



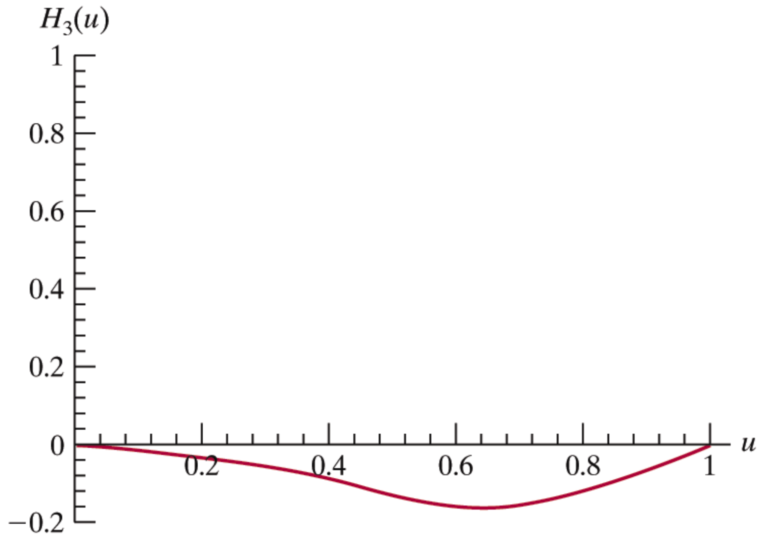
(a)



(b)



(c)



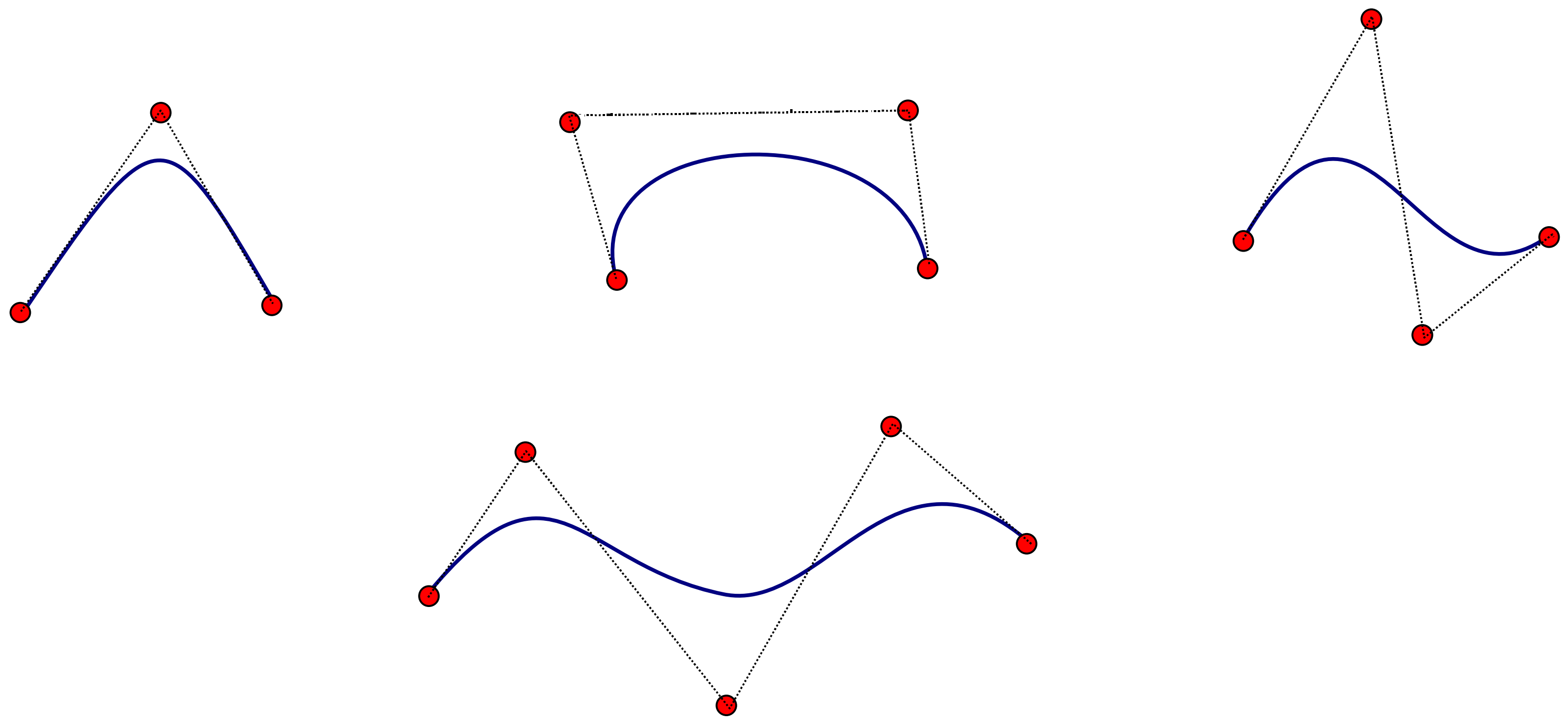
(d)

Hermite curves

- Segments are local. First order continuity
- Slopes at control points are required.
- Cardinal splines and Kochanek-Bartel splines approximate slopes from neighbor control points.

Bézier Curves

- A Bézier curve approximates any number of control points for a curve section (degree of the Bézier curve depends on the number of control points and their relative positions)



$$P(u)=\sum_{k=0}^{n} p_k \,\mathrm{BEZ}_{k,n}(u), \qquad 0\le u\le 1$$

$$\mathrm{BEZ}_{k,n}(u)=\binom{n}{k}u^k(1-u)^{n-k}, \qquad \binom{n}{k}=\frac{n!}{k!(n-k)!}$$

- The coordinates of the control points are blended using Bézier blending functions $\mathrm{BEZ}_{k,n}(u)$

- Polynomial degree of a Bézier curve is one less than the number of control points.
  3 points : parabola
  4 points : cubic curve
  5 points : fourth order curve

Cubic Bézier Curves

- Most graphics packages provide Cubic Béziers.

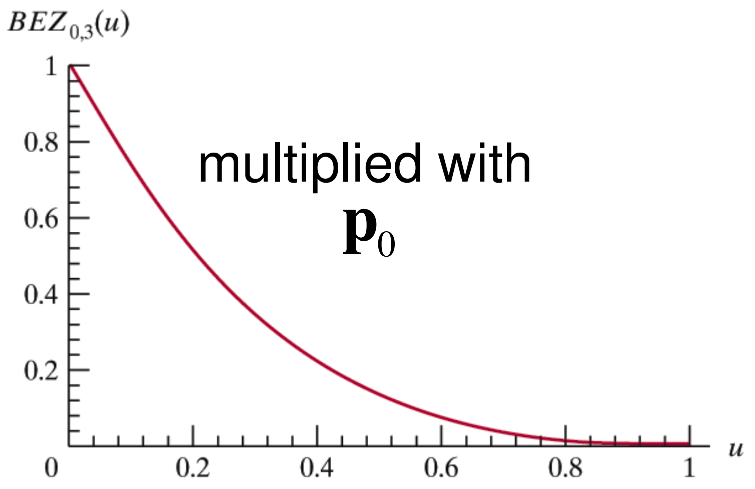
$$\mathrm{BEZ}_{0,3} = (1-u)^3 \qquad \mathrm{BEZ}_{1,3} = 3\mathrm{u}(1-u)^2$$

$$\mathrm{BEZ}_{2,3} = 3\mathrm{u}^2(1-u) \qquad \mathrm{BEZ}_{3,3} = u^3$$

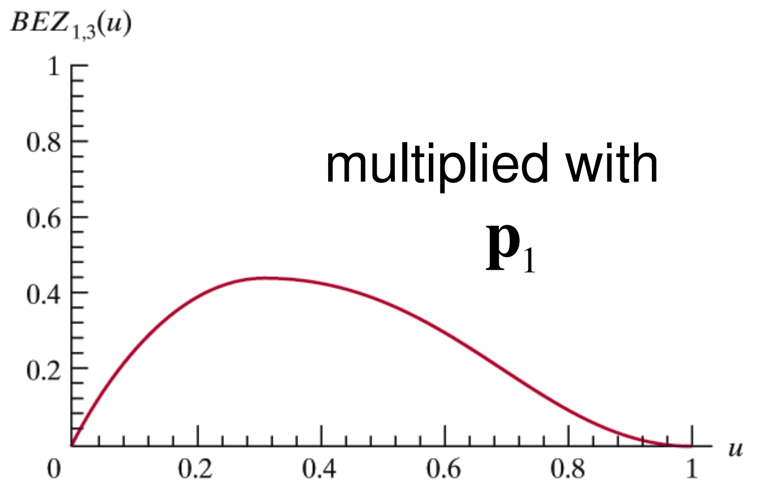
$$P(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \cdot M_{\mathrm{Bez}} \cdot \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

$$M_{\mathrm{Bez}} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

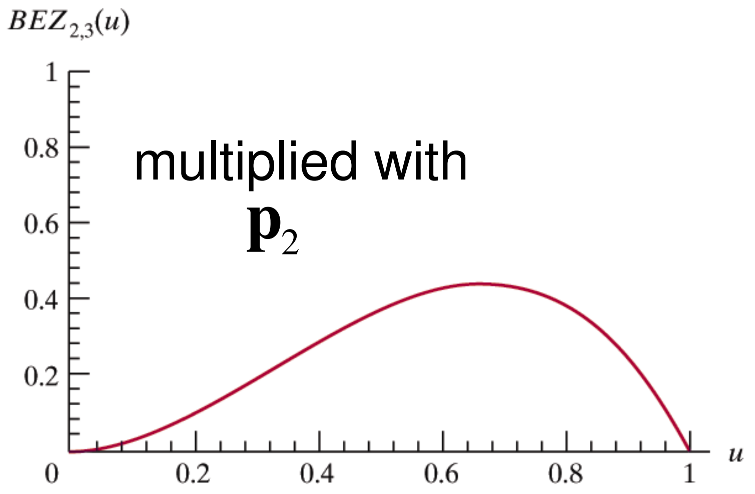
Cubic Bézier blending functions



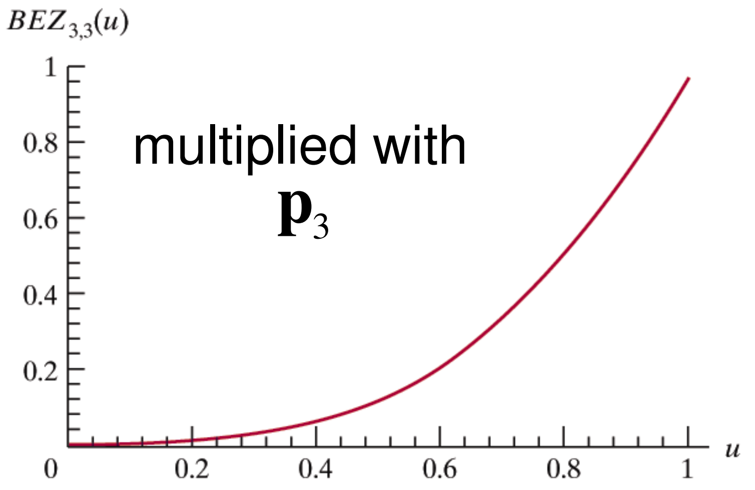
(a)



(b)



(c)



(d)

The four Bézier blending functions for cubic curves (n=3, i.e. 4 control pts.)

Properties of Bézier curves

- Passes through start and end points

$$P(0)=p_0, \qquad P(1)=p_n$$

- First derivates at start and end are:

$$P'(0)=-np_0+np_1$$
$$P'(1)=-np_{n-1}+np_n$$

- Lies in the convex hull

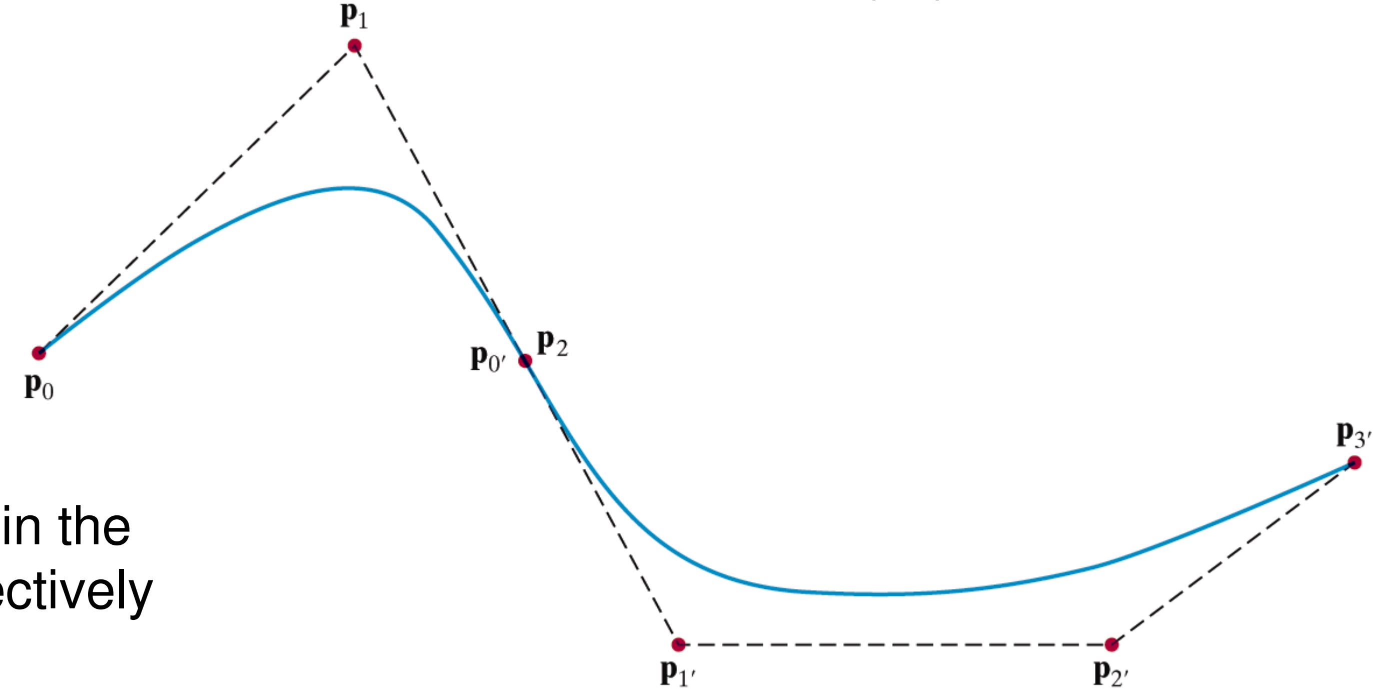
Joining Bézier curves

- Start and end points are same (C^0)
- Choose adjacent points to start and end in the same line (C^1)

$$p_{0'} = p_n$$

$$p_{1'} = p_n + \frac{n}{n'}(p_n - p_{n-1})$$

n and n' are the number of control points in the first and in the second curve segment respectively

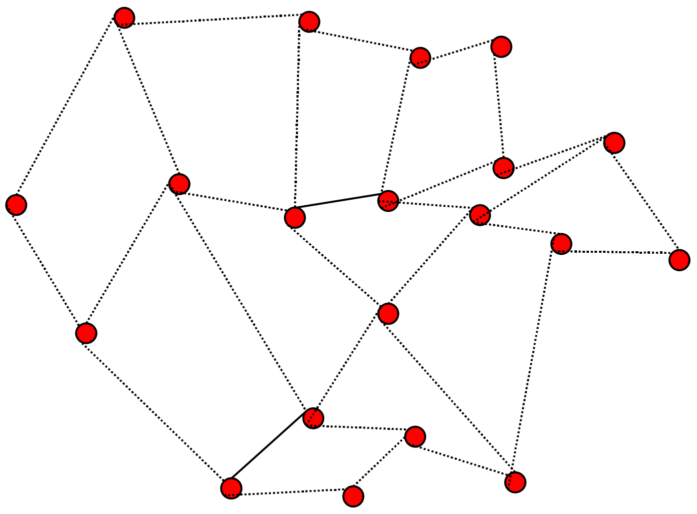


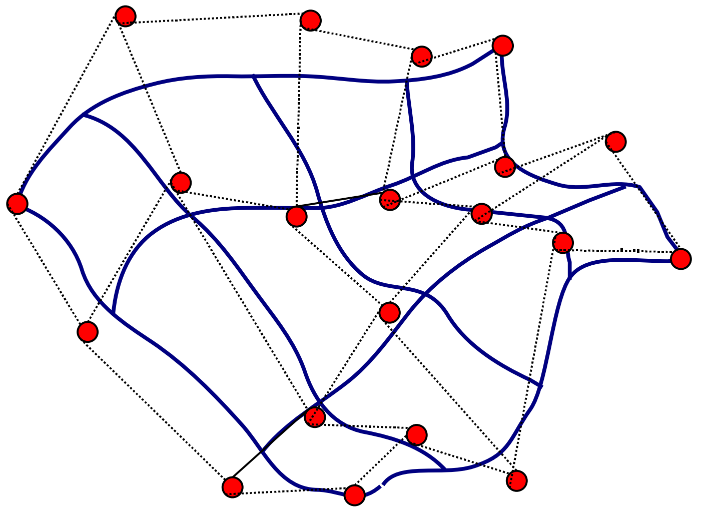
- C^2 continuity is not generally used in cubic Bézier curves. Because the information of the current segment will fix the first three points of the next curve segment

Bézier Surfaces

- Two sets of orthogonal Bézier curves are used.
- Cartesian product of Bézier blending functions:

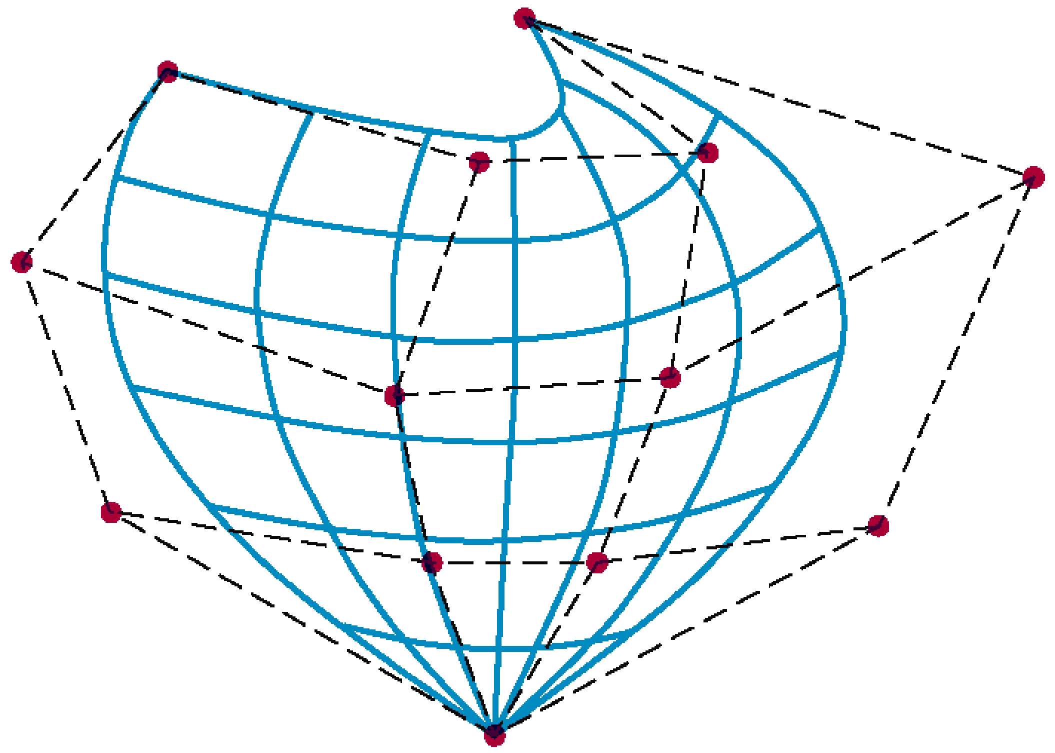
$$P(u,v) = \sum_{j=0}^{m} \sum_{k=0}^{n} p_{j,k} \mathrm{BEZ}_{j,m}(v) \mathrm{BEZ}_{k,n}(u) \qquad 0 \le u, v \le 1$$





Bézier Patches

- *A common form of approximating larger surfaces by tiling with cubic Bézier patches. m=n=3*
- 4 by 4 = 16 control points.



Cubic Bézier Surfaces

- Matrix form

$$P(u,v)=U\cdot M_{\text{Bez}}\cdot P\cdot M_{\text{Bez}}^{T}\cdot T^{T}=$$

$$\begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix}\cdot\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\cdot\begin{bmatrix} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \\ p_{2,0} & p_{2,1} & p_{2,2} & p_{2,3} \\ p_{3,0} & p_{3,1} & p_{3,2} & p_{3,3} \end{bmatrix}\cdot\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}\cdot\begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

- Joining patches:
  similar to curves. C^0, C^1 can be established by choosing control points accordingly.

Displaying Curves and Surfaces

- Horner's rule: less number of operations for calculating polynoms.

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x$$

$$x(u) = ((a_x u + b_x) u + c_x) u + d_x$$

- Forward-difference calculations:
  Incremental calculation of the next value.
  - Linear case and using subintervals of fixed size to divide u:

$$u_{k+1} = u_k + \delta, \qquad k = 0,1,2,\ldots \qquad u_0 = 0$$

$$x_k = a_x u_k + b_x \qquad x_{k+1} = a_x(u_k + \delta) + b_x$$

$$x_{k+1} = x_k + x \qquad \boxed{x = a_x \delta} \longrightarrow \text{constant}$$

Forward-difference for cubic-splines

- Cubic equations

$$x_k = a_x u_k^3 + b_x u_k^2 + c_x u_k + d_x \qquad x_{k+1} = a_x (u_k + \delta)^3 + b_x (u_k + \delta)^2 + c_x (u_k + \delta) + d_x$$

$$x_k = 3\mathrm{a}_x \delta u_k^2 + (3\mathrm{a}_x \delta^2 + 2\mathrm{b}_x \delta) u_k + (a_x \delta^3 + b_x \delta^2 + c_x \delta)$$

$$x_{k+1} = x_k + {}_2 x_k \qquad {}_2 x_k = 6\mathrm{a}_x \delta^2 u_k + 6\mathrm{a}_x \delta^3 + 2\mathrm{b}_x \delta^2$$

$${}_2 x_{k+1} = {}_2 x_k + {}_3 x_k \qquad {}_3 x_k = 6\mathrm{a}_x \delta^3$$

$$x_0 = d_x$$

$$x_0 = a_x \delta^3 + b_x \delta^2 + c_x \delta$$

$${}_2 x_0 = 6\mathrm{a}_x \delta^3 + 2\mathrm{b}_x \delta^2$$

Once we compute these initial values, the calculation for next x-coordinate position takes only three additions.

Example

$$x_0 = d_x$$

$$\Delta x_0 = a_x \delta^3 + b_x \delta^2 + c_x \delta$$

$$\Delta_2 x_0 = 6\mathrm{a}_x \delta^3 + 2\mathrm{b}_x \delta^2$$

$$\Delta_3 x_k = 6\mathrm{a}_x \delta^3$$

- Example:

$(a_x, b_x, c_x, d_x) = (1,2,3,4)$, $\delta = 0.1$

$$\Delta_3 x_k = 6 \cdot 1 \cdot \delta^3 = 0.006$$

x	Δx	$\Delta_2 x$
4.000	0.321	0.046
4.321	0.367	0.052
4.688	0.419	0.058
5.107	0.477	0.064
5.584	0.541	0.070
6.125	0.611	0.076
6.736	0.687	0.082
7.423	0.769	0.088
8.192	0.857	0.094
9.049	0.951	0.100

OpenGL Bézier-Spline Curve Functions

- glMap1*() to specify control points
- glEnable (GL_MAP1_VERTEX_3)
  - Activate curve generation routines
- glDisable (GL_MAP1_VERTEX_3)
  - Deactivate curve generation routines
- glEvalCoord1* (uValue)
  - Generates the point coordinate at the uValue
  - It actually generates a glVertex3 function!!

Example

```
GLfloat ctrlPts [4] [3] = {{0.0,1.0,2.0},……};
glMap1f (GL_MAP1_VERTEX_3,0.0,1.0,3,4,&ctrlPts[0][0]);
glEnable (GL_MAP1_VERTEX_3);
Glint k;
glBegin (GL_LINE_STRIP);
        for (k=0;k<=50;k++)
        glEvalCoord1f(GLfloat (k) / 50.0);
glEnd ();
```

Generating uniformly spaced *u* values

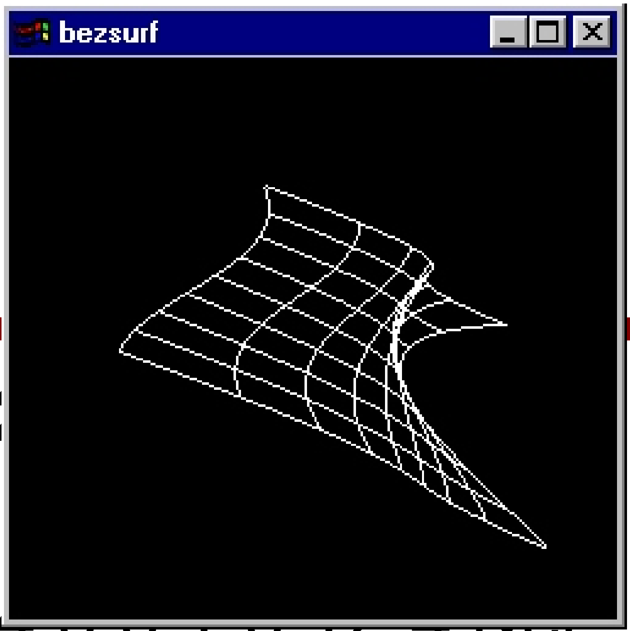
We may replace the glBegin(), inner for loop, and glEnd() with:

```
glMapGrid1f (50, 0.0, 1.0);
glEvalMesh1 (GL_LINE, 0, 50);
```

OpenGL Bézier-Spline Surface Functions

- glMap2* () to specify control points
- glEnable (GL_MAP2_VERTEX_3)
  - Activate curve generation routines
- glDisable (GL_MAP2_VERTEX_3)
  - Deactivate curve generation routines
- glEvalCoord2* (uValue, vValue)
  - Generates the point coordinate at the uValue, vValue
  - This also generates a glVertex3 function.

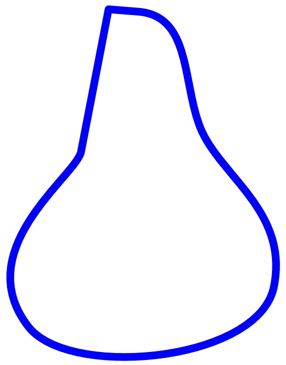
Example



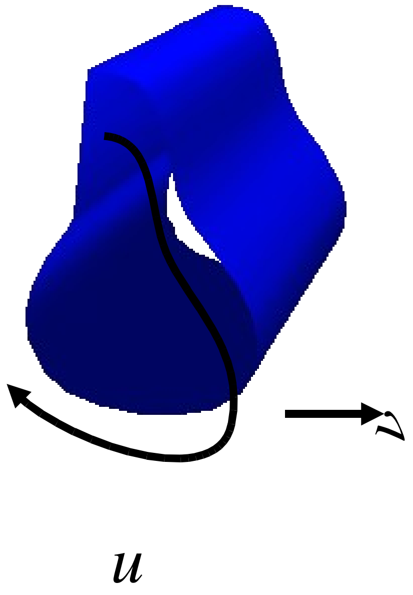
```
GLfloat ctrlPts [4] [4] [3] = {{{0.0,1.0,

glMap2f
   (GL_MAP2_VERTEX_3,0.0,1.0,3,4,0.0,1.0,12,4,&ct
   rlPts[0][0][0]);

glEnable (GL_MAP2_VERTEX_3);


glMapGrid2f (40,0.0,1.0,40,0.0,1.0);

glEvalMesh2 (GL_LINE,0,40,0,40);

// GL_POINT and GL_FILL is also available with
   glEvalMesh2()
```

Sweep Representations

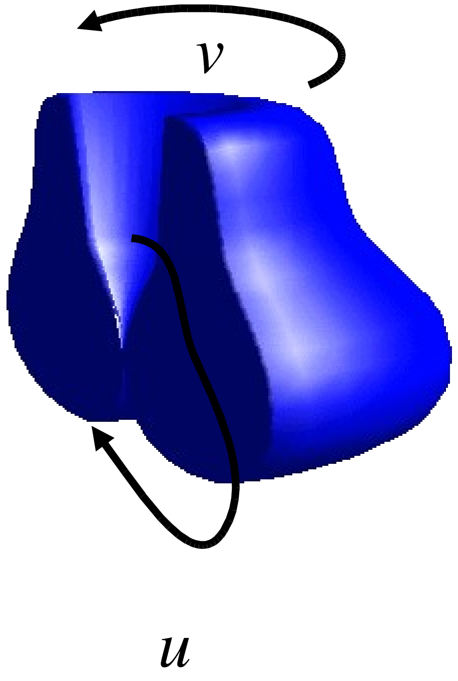
- Use reflections, translations and rotations to construct new shapes.



P(u)



Translational Sweep



Rotational Sweep

Translational Sweep

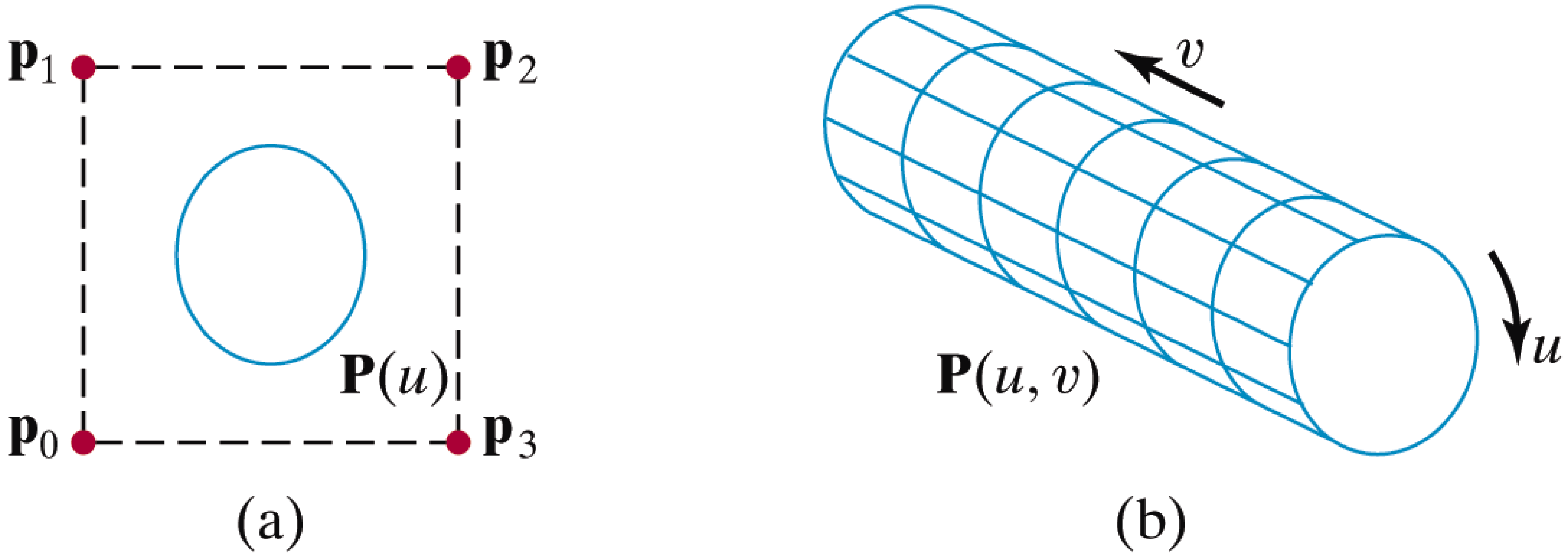


Figure 8-55

Constructing a solid with a translational sweep. Translating the control points of the periodic spline curve in (a) generates the solid shown in (b), whose surface can be described with the point function $\mathbf{P}(u,v)$.

Rotational Sweep

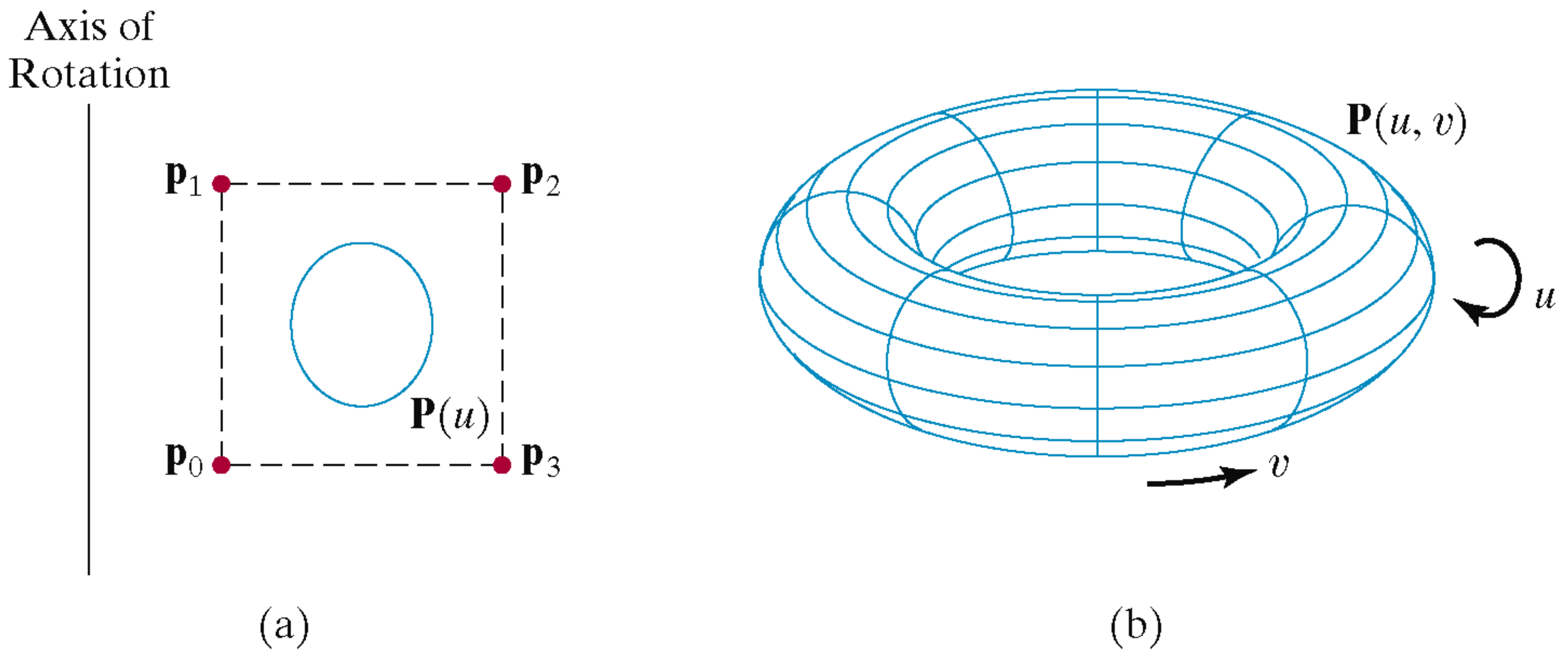


Figure 8-56

Constructing a solid with a rotational sweep. Rotating the control points of the periodic spline curve in (a) about the given rotation axis generates the solid shown in (b), whose surface can be described with the point function $\mathbf{P}(u,v)$.

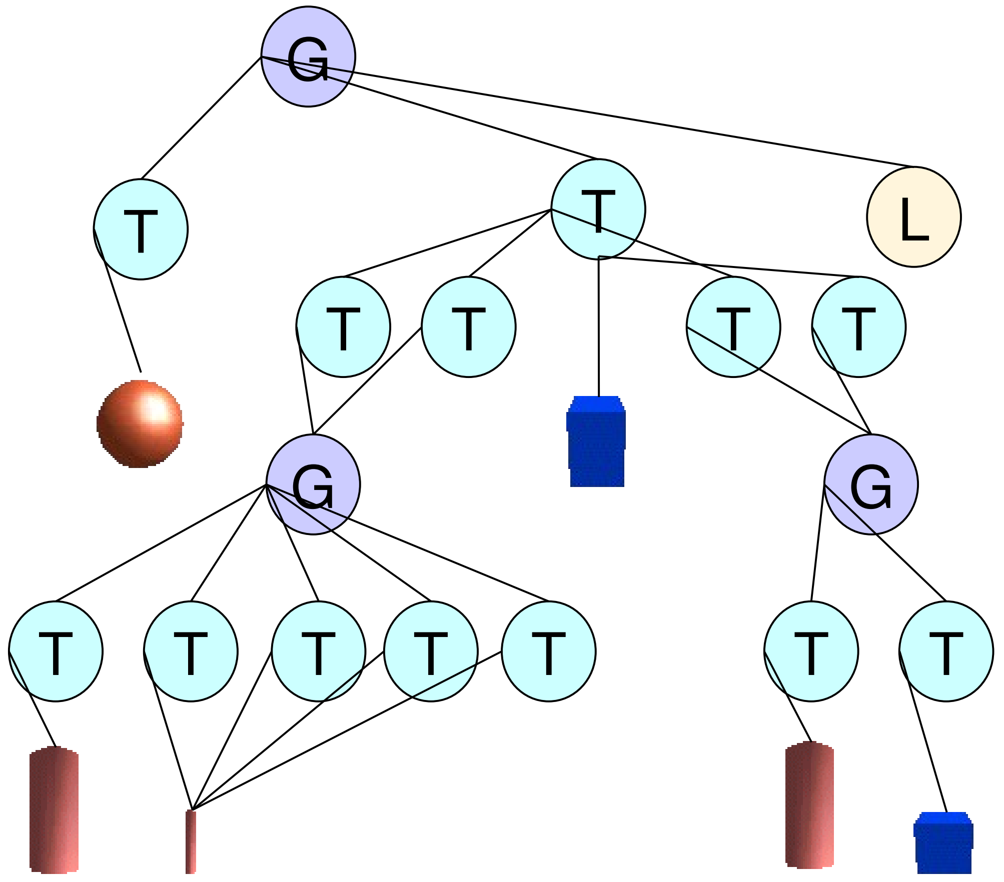
Hierarchical Models

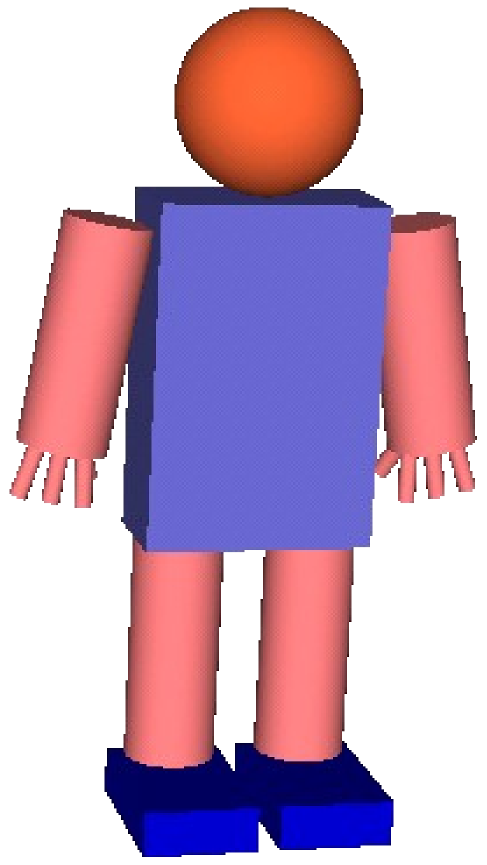
- Combine smaller/simpler shapes to construct complex objects and scenes.
- Stored in trees or similar data structures
- Operations are based on traversal of the tree
- Keeping information like bounding boxes in tree nodes accelarate the operations.

Scene Graphs

- DAG's (Directed Acyclic Graphs) to represent scenes and complex objects.
- Nodes: Grouping nodes, Transform nodes, Level Of Detail nodes, Light Source nodes, Attribute nodes, State nodes. Leaves: Object geometric descriptions.
- Why not tree but DAG?
- Available libraries: i.e. www.openscenegraph.org
  - Java3D is also based on Scene Graph Model
- Efficient display of objects, picking objects, state change and animations.

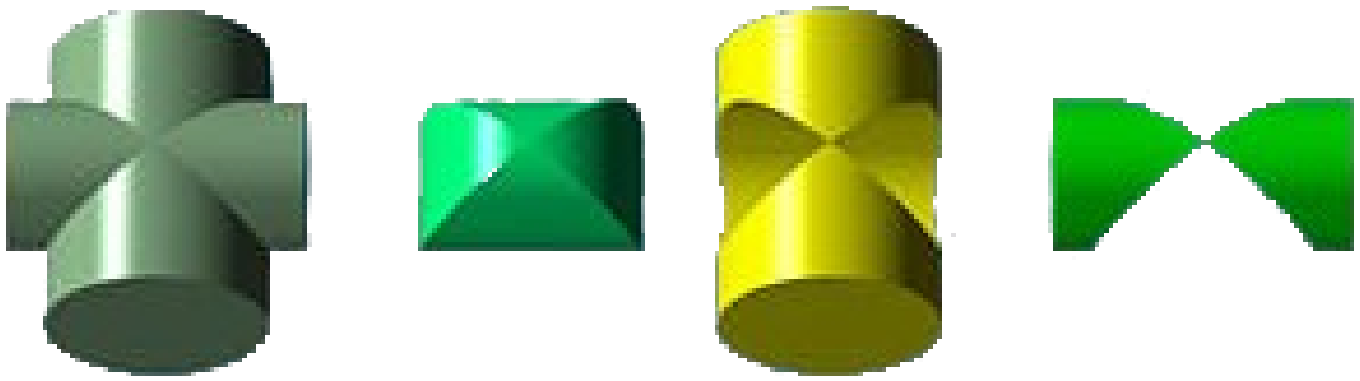
Scene Graph Representation





Constructive Solid Geometry

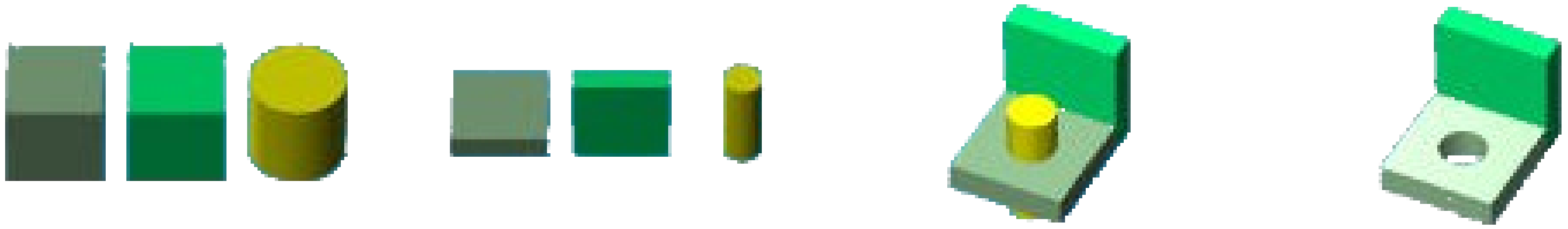
- Combine multiple shapes with set operations (intersection, union, deletion) to construct new shapes.



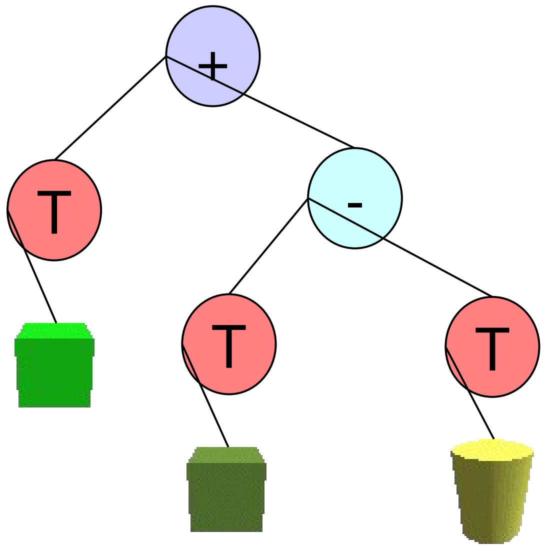
$A \cup B$ $A \cap B$ $A - B$ $B - A$

CSG Tree Representation

- Set operations and transformations combined:



- `union(transA(box),diff(transB(box),transC(cylinder)))`

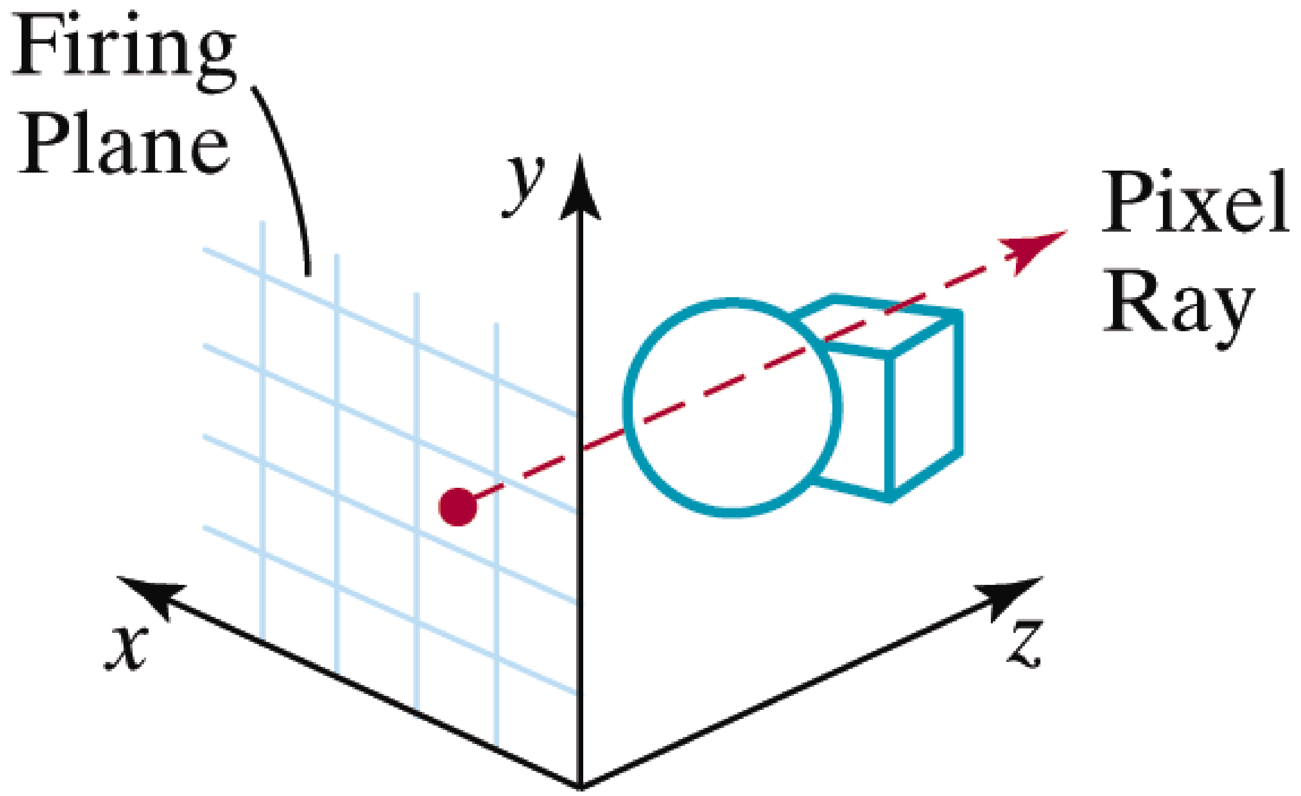


A CSG Tree

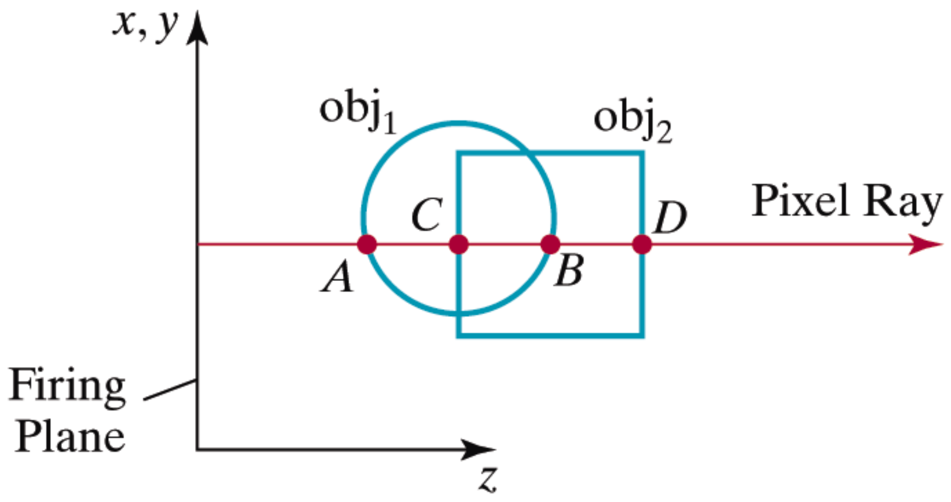
Implementing CSG

- Ray casting methods are used for rendering and finding properties of volumes constructed with this method.
- Parallel lines emanating from the *xy* plane (firing plane) along the *z* direction are intersected with the objects
- The intersection points are sorted according to the distance form the firing plane
- Based on the operation the extents (i.e., the surface limits) of the constructed object can be found.

Ray Casting



Ray Casting

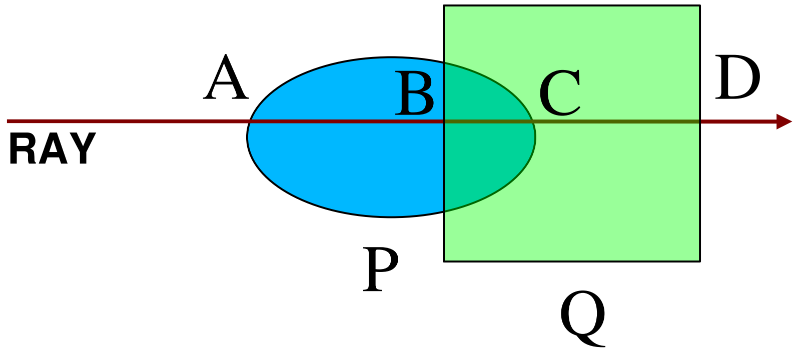


Operation	Surface Limits
Union	A, D
Intersection	C, B
Difference ($obj_2 - obj_1$)	B, D

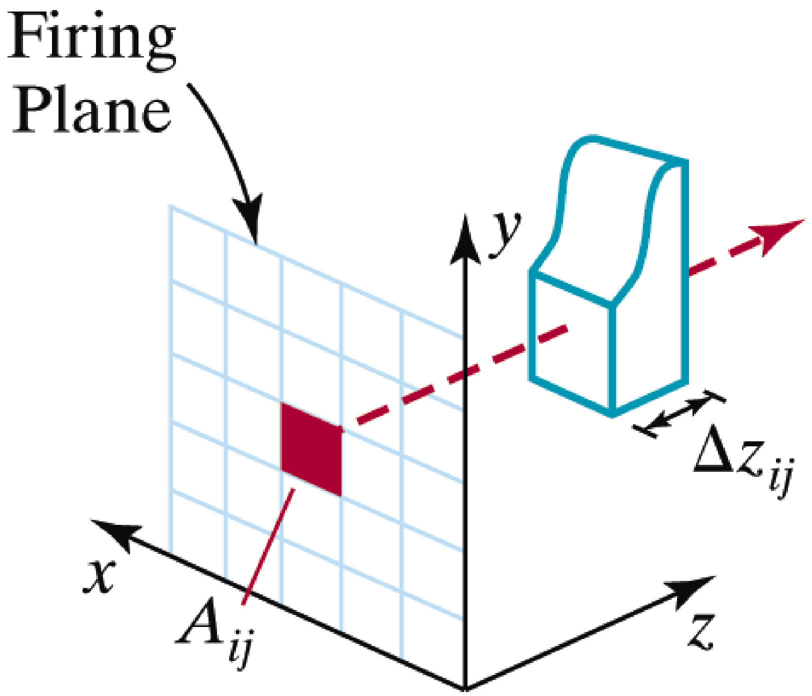
Implementation Details

- Simply +1 for outside→inside, -1 for inside→outside transition. Positives are solid.

	A	B	C	D
P ∪ Q	1	2	1	0
P ∩ Q	0	1	0	0
P - Q	1	0	-1	0
Q - P	-1	0	1	0



Calculating the Volume



Volume along the ray:

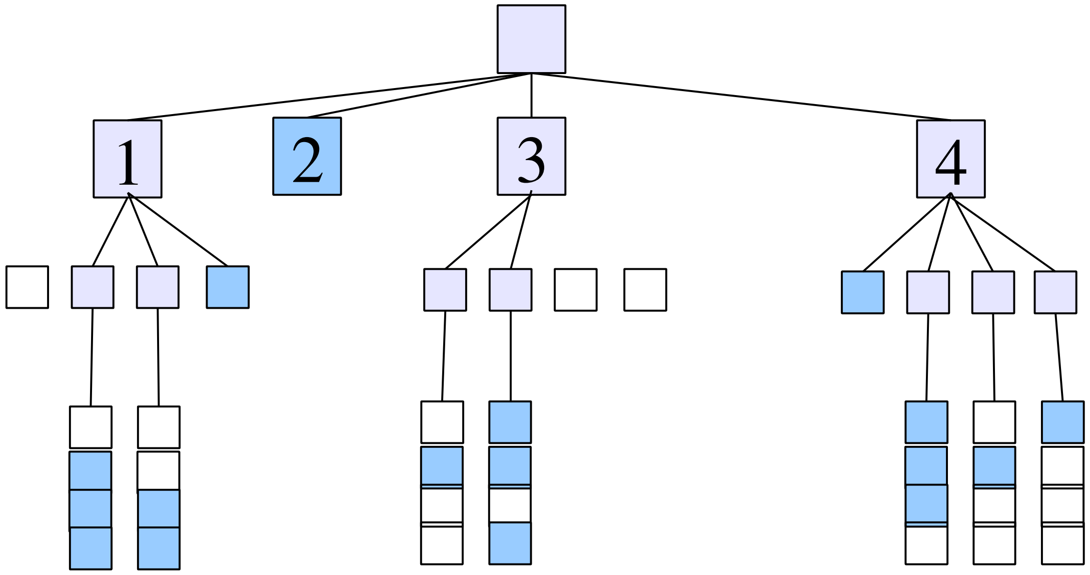
$$V_{ij} \approx A_{ij}\ z_{ij}$$

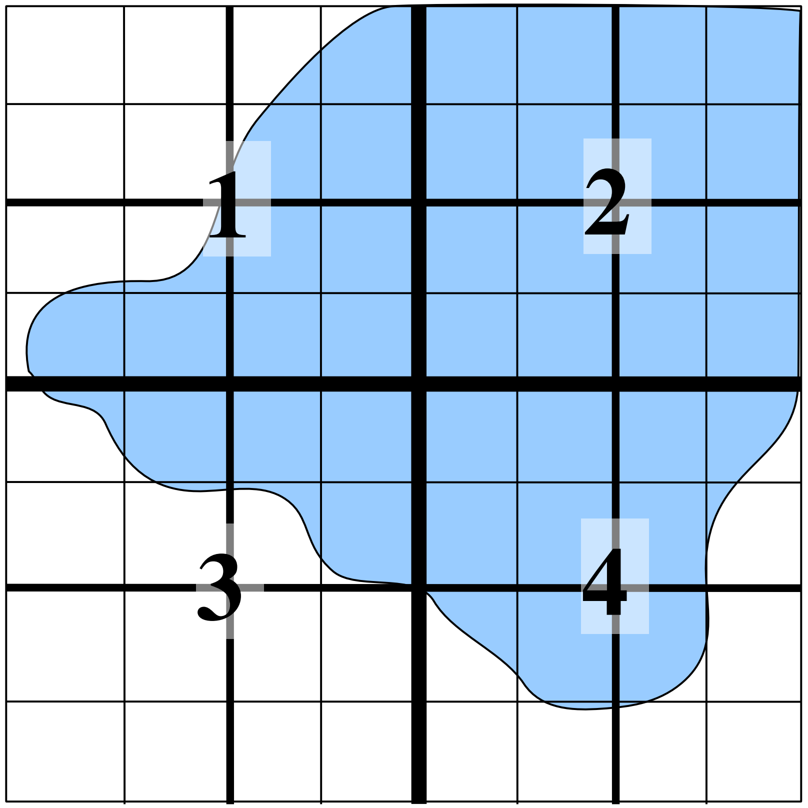
Total Volume:

$$V \approx \sum_{i,j} V_{ij}$$

Octrees

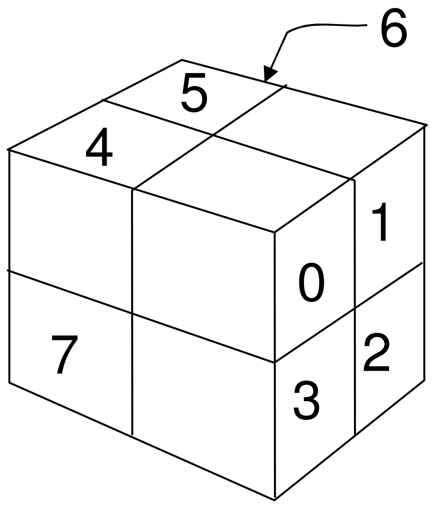
- Divide a volume in equal binary partitions in all dimensions recursively to represent solid object volumes. Combining leaf cubes gives the volume.
- 2D: quadtree





- 2D: quadtree; 3D: octree
- Volume data: Medical data like Magnetic Resonance. Geographical info (minerals etc.)
- 2D: Pixel ; 3D: voxel.
- Volumes consisting of large continous subvolumes with properties. Volumes with many wholes, spaces. Surface information is not sufficient or tracktable.
- Keeping all volume in terms of voxels, too expensive: space and processor.
  - Therefore, homogeneous regions of the space can be represented by larger spatial components

- 8 elements at each node.
- If volume completely resides in a cube, it is not further divided: leaf node
- Otherwise nodes are recursively subdivided.



- Extent of a tree node is the extent of the cube it defines.
- Surfaces can be extracted by traversing the leaves with geometrical adjacency.

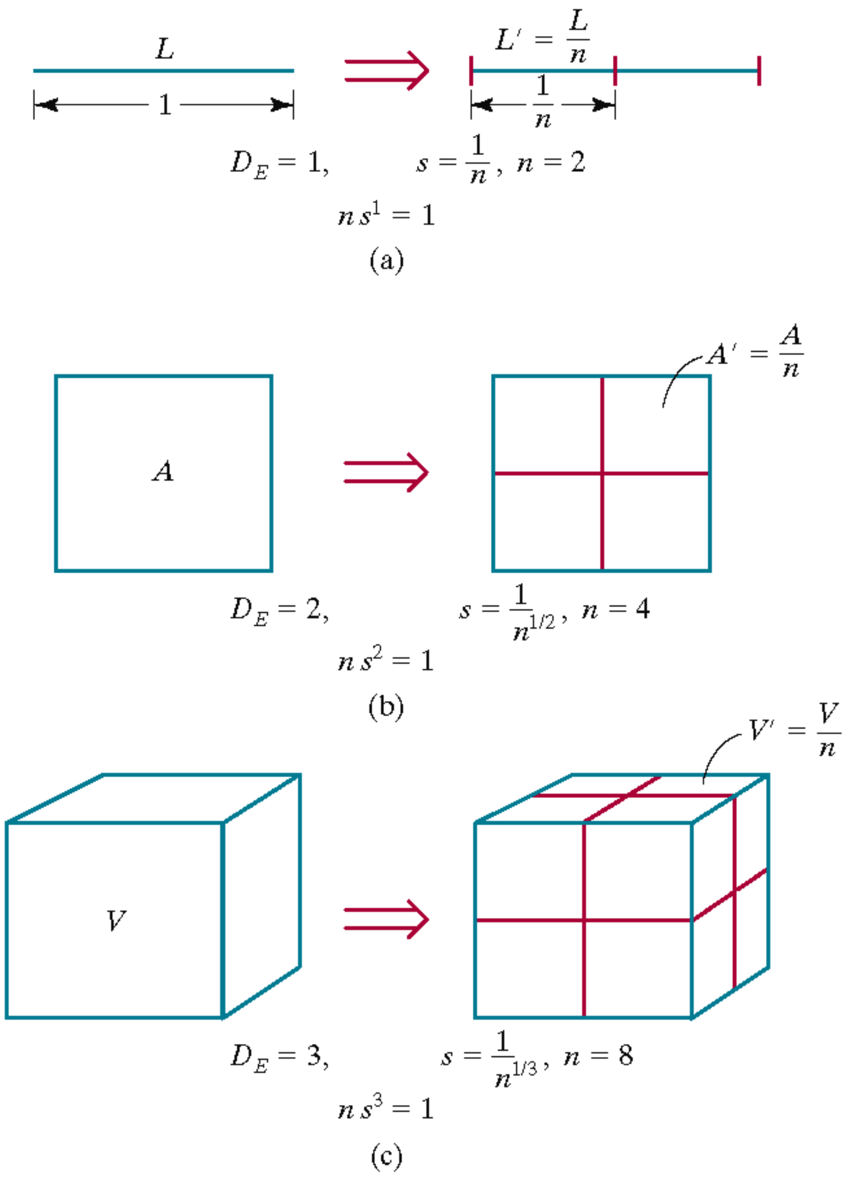
Fractal Geometry Methods

- Synthetic objects: regular, known dimension
- Natural objects: recursive (self repeating), the higher the precision, the higher the details you get.
- Example: tree branches, terrains, textures.
- Classification:
  - Self-similar: same scaling parameter *s* is used in all dimensions, scaled-down shape is similar to original
  - Self-affine: self similar with different scaling parameters and transformations. Statistical when random parameters are involved.

Fractal Dimension

- Fractal dimension:
  - Amount of variation of a self similar object. Denoted as *D*.
  - Fragmentation, roughness of the object.
- The fractal dimension of a self-similar fractal with a single scaling factor s is obtained using ideas from subdivision of a Euclidean object.

Fractal Dimension



L
$L' = \frac{L}{n}$
1
$\frac{1}{n}$
$D_E = 1, \quad s = \frac{1}{n}, \; n = 2$
$n s^1 = 1$
(a)
A
$A' = \frac{A}{n}$
$D_E = 2, \quad s = \frac{1}{n^{1/2}}, \; n = 4$
$n s^2 = 1$
(b)
V
$V' = \frac{V}{n}$
$D_E = 3, \quad s = \frac{1}{n^{1/3}}, \; n = 8$
$n s^3 = 1$
(c)

The relationship between the number of subparts and the scaling factor is

$$n \cdot s^{D_E} = 1$$

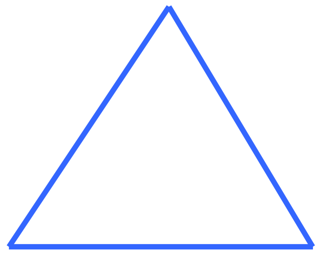
where D_E is the Euclidean dimension.

We can define the fractal dimension similarly with number of subparts n and a given scaling factor s.

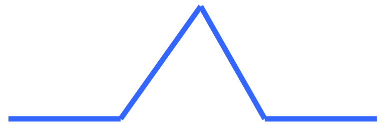
$$n \cdot s^{D} = 1$$

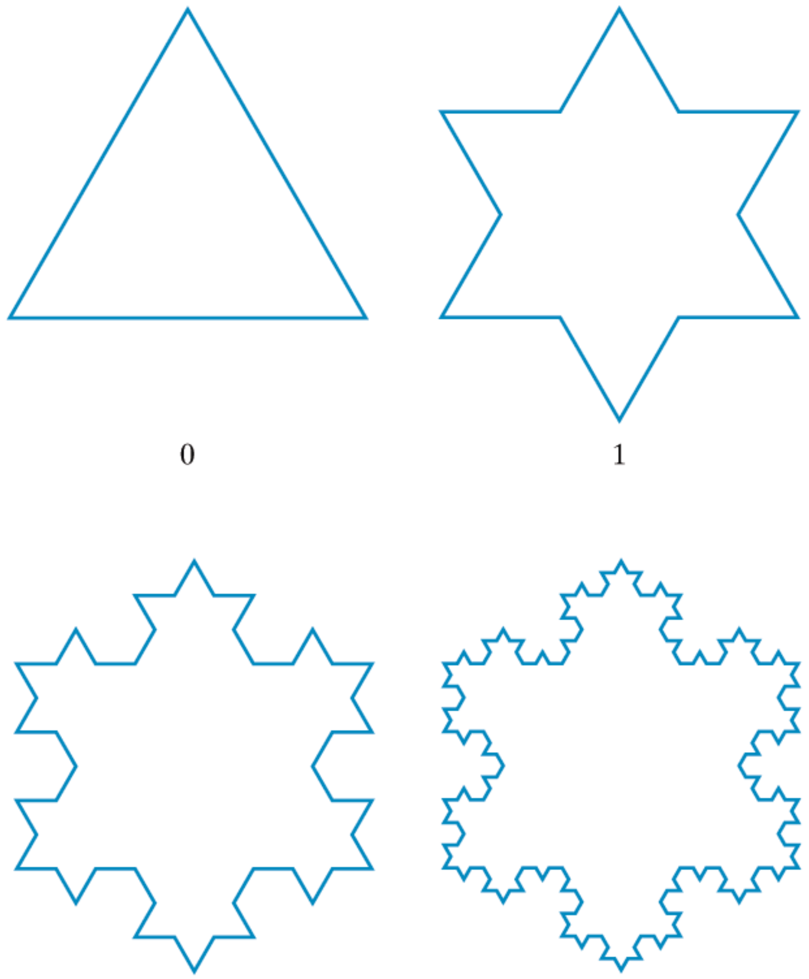
Koch curve

Initiator:



Generator:



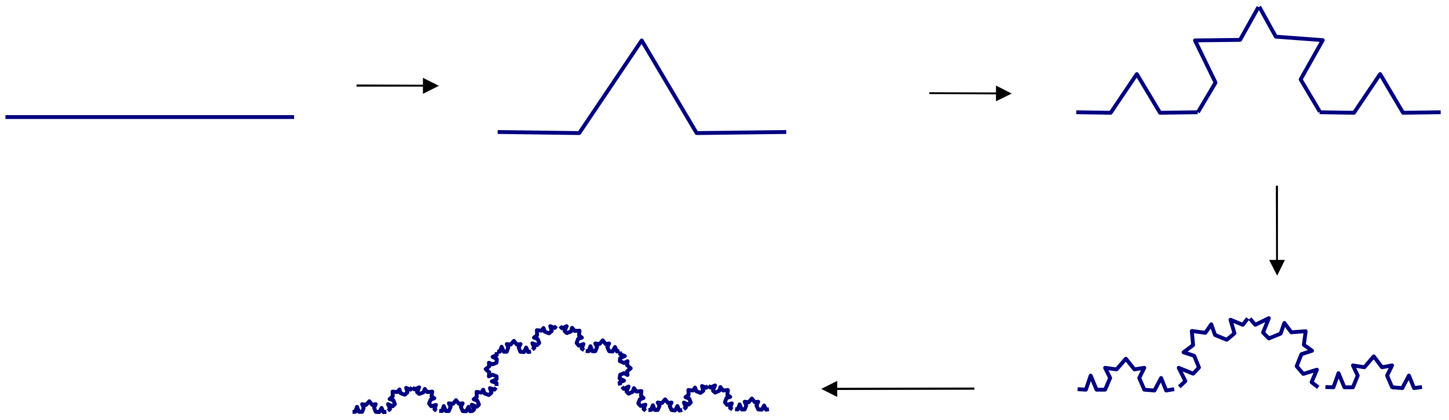


The fractal dimension of the Koch curve

$$n s^D = 1$$

$$D = \frac{\ln n}{\ln(1/s)}$$

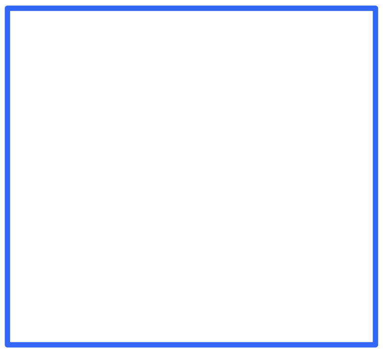
n : *number of pieces* s : *scaling factor*



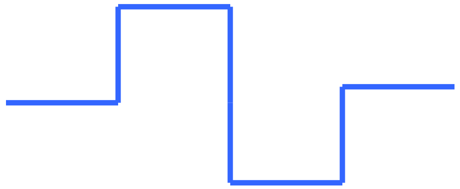
$n = 4 \qquad s = 1/3 \qquad D = \frac{\ln 4}{\ln 1/(1/3)} = 1.2619$

Exercise

- What is the fractal dimension of the following shape?



Initiator



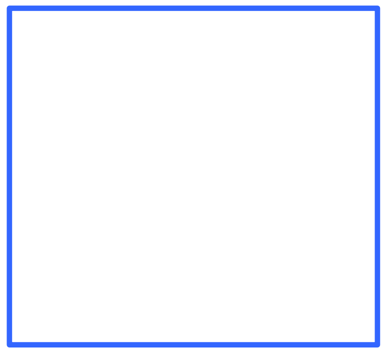
Generator

Each edge of the square is replaced by the generator recursively.

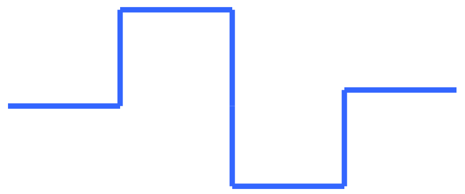
What is n? What is s?

Exercise

- What is the fractal dimension of the following shape?



Initiator



Generator

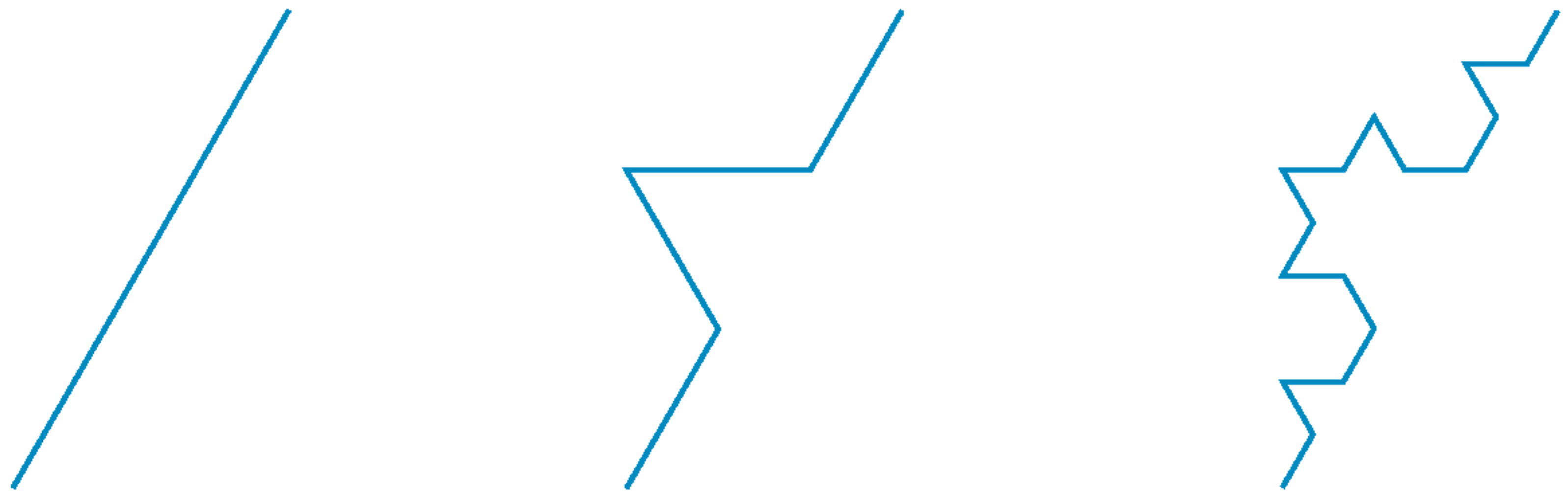
Each edge of the square is replaced by the generator recursively.

$n = 8, \quad s = 1/4, \ D = 1.5$

Length of a fractal curve

- Increases at each subdivision. Infinite length at infinite detail.

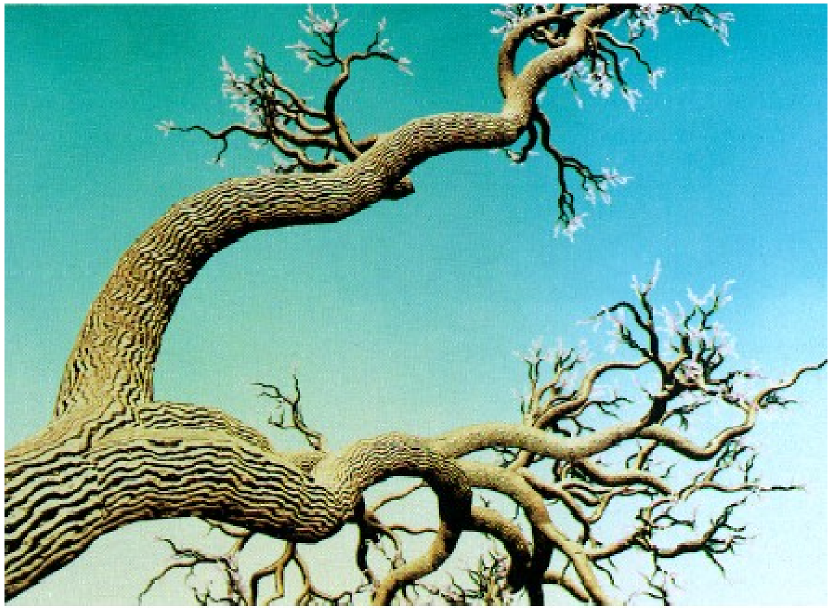
Segment Length $= 1$ Segment Length $= \frac{1}{3}$ Segment Length $= \frac{1}{9}$



Length $= 1$ Length $= \frac{4}{3}$ Length $= \frac{16}{9}$

Random Variation

- Using a probability distribution function we can introduce small variations during the subdivision of the object for more realism.



Images from a 1986 paper.

Random Mid-point Variation

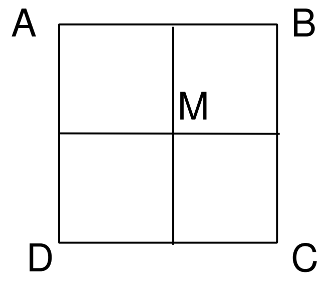
- Find the midpoint of an edge A-B. Add a random factor and divide the edge in two as: A-M, M-A at each step.
- Usefull for height maps, clouds, plants.
- 2D:

$$x_m = (x_A + x_B)/2$$
$$y_m = (y_A + y_B)/2 + r$$

r is a random number selected from Gaussian distribution with mean 0 and variance that depends on D and the length between end points

- 3D: For corners of a square: A, B, C, D

$$Z_{AB} = (Z_A + Z_B)/2 + r, \qquad Z_{BC} = (Z_B + Z_C)/2 + r,$$
$$Z_{CD} = (Z_C + Z_D)/2 + r, \qquad Z_{DA} = (Z_D + Z_A)/2 + r,$$
$$Z_M = (Z_{AB} + Z_{BC} + Z_{CD} + Z_{DA})/4 + r,$$



Random Mid-point Variation