Prof. Reuven Aviv
Dept. of Computer Science
Tel Hai Academic College
Computer Graphics

1. Intro; Windows and Viewports

Slides adapted from: F.S. Hill, S.M. Kelley: Computer Graphics with OpenGL

This course

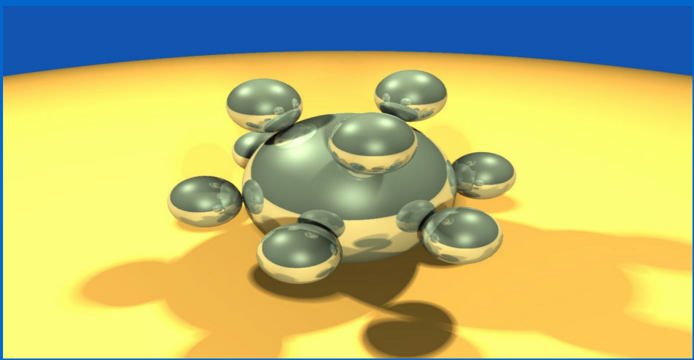
- 4 hrs Monday 14:30 – 18:00
- Theory & Developing programs
- Math: Algebra (vectors, matrices)
- Diff Calculus II
- Assignments:
- Problem Sets (3/4) – 20%
- Programming Assignments (3/4) – 20%
- Presentation (OpenGL) – 10%
- Exam I – 15%
- Final Exam – 35%

This course (2): Topics

- 1. Windows Vieports
- 2. Vector Mathematics
- 3. OPENGL Drawing objects (STUDENTS)
- 4. Transformations
- 5. 3D Viewing
- 6. OPENGL Viewing (STUDENTS)
- 7. Eaxm; Mesh Modeling
- 8. Rendering faces, realism
- 9. OPENGL Lightning, Texture
- 10. Raster Operations
- 11. Curve and Surface Design
- 12. OPENGL Curve and Surface Design
- 13. ray Tracing

What is Computer Graphics?

- Pictures generated by a computer
  - Example: a ray-traced picture with shadows.



Computer Graphics Tools

- HW/SW.
  - video monitors, graphics cards, printers
  - input devices
- Software tools: Graphics routines
  - Window management, dialog, …
  - set up a camera in 3D coordinate system and take snapshots of objects
  - Device Independent Libraries (OpenGL)
- *What is the diff between Computer Graphics and Image Processing?*

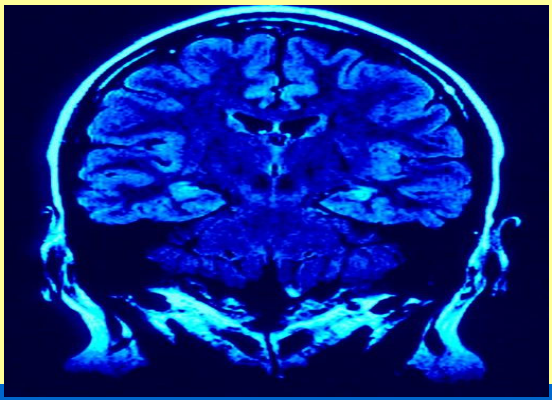
Computer Graphics and Image Processing

- Computer graphics create pictures and images based on some model.
- Image processing improves or alters images
  - remove noise, enhance contrast, sharpen…
  - search for certain features in an image, and highlight them…
- *Name some applications of computer graphics*

Application: Movies

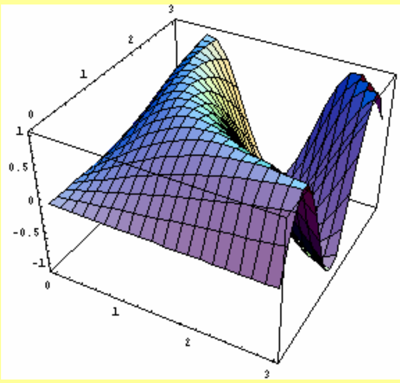
Application: Volume Visualization

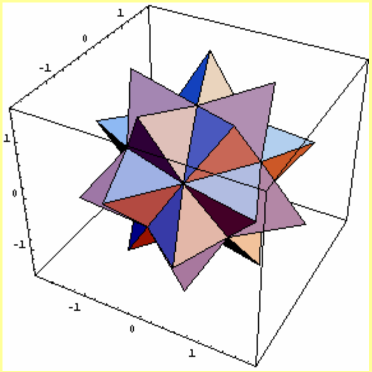
- **Areas of different colors immediately inform a physician about the health of each part of the brain.**



Application: Displaying Mathematical Functions

- **E.g., Mathematica®**



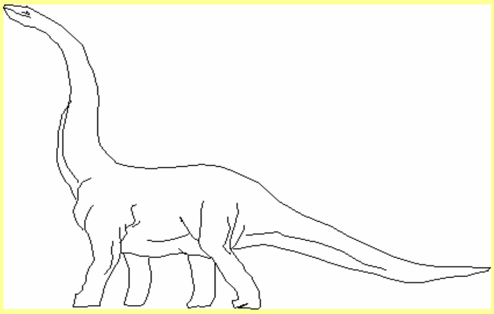


Models

- *What's the diff between a model and its picture?*
- Model consists of primitives
  - Points, lines, polylines, text, regions
- 3D bodies modeled by primitives
- triangles → Mesh → surface
  - Location, orientation
- Attributes: **e.g. color, thickness**
- **Light sources:**
  - **Direct, ambient**
- **More attributes: Reflectance, transparent**

Modeling by Polylines

- polyline: connected sequence of straight lines.
- (x_0, y_0), (x_1, y_1), (x_2, y_2), ..., (x_n, y_n).
- The model must be processed
  - Translated, rotated, scaled etc..



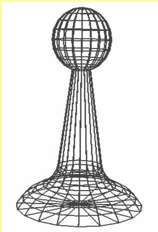
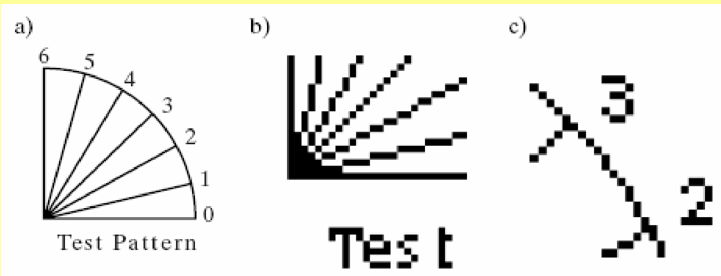


Image Processing Example: "Jaggies"

- Any close-up version an image will show that it is composed of pixels rather than lines. Thus the lines also appear jagged (the Jaggies).



Modeling and Viewing

- **We want to separate the coordinates we use in a program to *describe* the geometrical object from the coordinates we use to size and position the *pictures* of the objects on the display.**
- *Why?*
- **Description is usually referred to as a** modeling **task, and displaying pictures as a** viewing **task.**

World Coordinates

- **The coordinates by which objects are described are called** world coordinates
- **the numbers used for x and y (and z) are those in the world, where the objects are defined.**
- *Which part of the world we wish to take a picture of?*

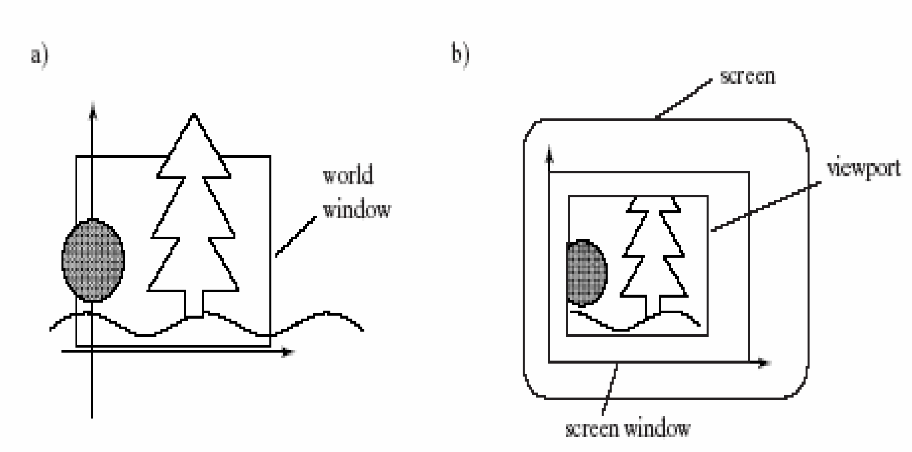
World Window, Clipping

- **We define a rectangular world window in these world coordinates.**
- **The world window specifies which part of the world should be drawn: what inside the window should be drawn, what lies outside should be clipped away and not drawn.**
- **OpenGL does the clipping automatically**
- *Where on the screen the picture will be drawn?*

Viewport

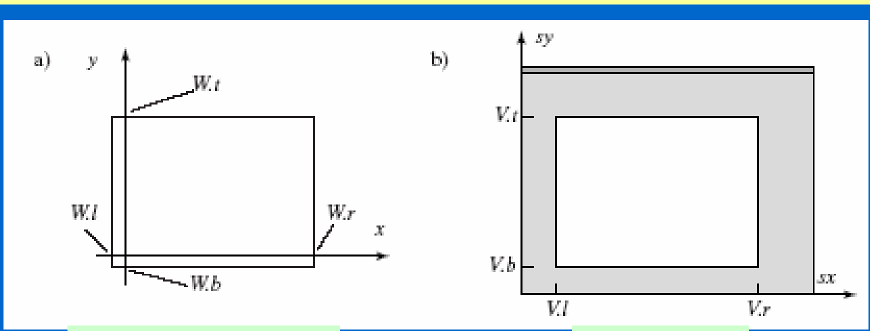
- **we define a rectangular viewport in the screen window on the display.**
- **A mapping between the world window and the viewport is established by OpenGL.**
- *What kind of transformations are included in this mapping?*
- **Scaling and translation**
- **The objects inside the world window are automatically transformed in sizes and locations to be inside the viewport (in** screen coordinates, **which are pixel coordinates on the display).**

Windows and Viewport



Window to Viewport transformation

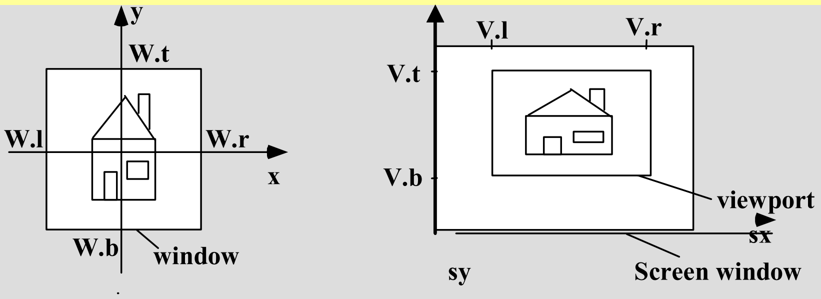
- **Window is described by its boundaries**
  - **left, top, right, bottom values, w.l, w.t, w.r, w.b.**
- ***How the viewport is described?***
- **v.l, v.t, v.r, v.b, in screen window coordinates.**



World window **Viewport**

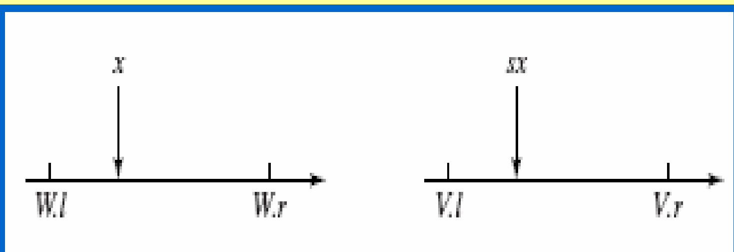
Window to Viewport transformation

- **Transformation: aligned rect. to aligned rect.**
  - **If the aspect ratios of the 2 rectangles are not the same, distortion will result.**
- *What do we require from the transformation?*



Window to Viewport transformation

- **Proportionality requirement:**
- **example, if x is ¼ of the way between *left* and *right*, then the screen x (s_x) should be ¼ of the way between the left and right viewport boundaries.**
- *What kind of transformation this is?*



Window-to-Viewport Transformation (2)

- **The mapping must be linear.**
  - $sx = Ax + C,\ sy = B\,y + D$
  - **We require**
  - $(sx - V.l)/(V.r - V.l) = (x - W.l)/(W.r - W.l)$
  - $(sy - V.b)/(V.t - V.b) = (y - W.b)/(W.t - W.b)$

Window-to-Viewport Transformation (3)

- **Result**
- $sx = A\,x + C,\ sy = B\,y + D$, **with**

$$A = \frac{V.r - V.l}{W.r - W.l},\ C = V.l - A \cdot W.l$$

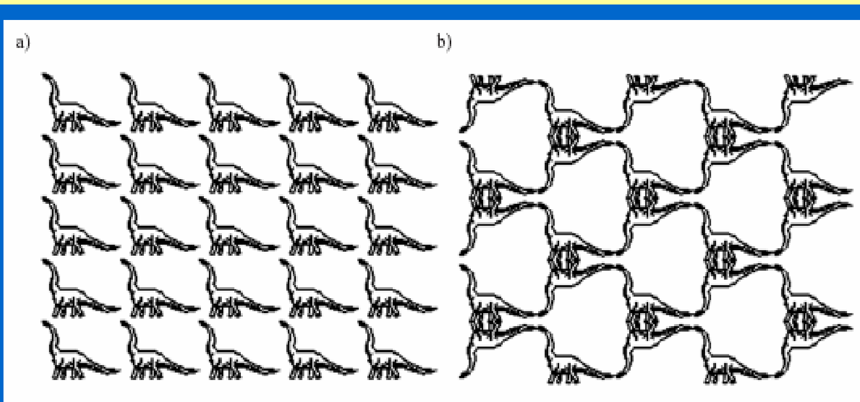
$$B = \frac{V.t - V.b}{W.t - W.b},\ D = V.b - B \cdot W.b$$

OpenGL Functions To do the Mapping

- **Defining the World window:**
  - **void *gluOrtho2D*(GLdouble *left*, GLdouble *right*, GLdouble *bottom*, GLdouble *top*);**
- **Viewport:**
  - **void *glViewport*(GLint *x*, GLint *y*, GLint *width*, GLint *height*);**
- **All objects defined in the modeling step are transformed by this transformation**
- **End of the *Graphics pipeline***

Application: Tiling

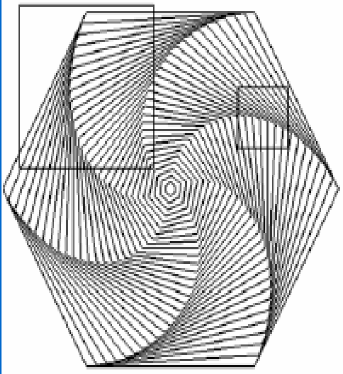
- *How can we create these pictures?*



a. Shift the Viewport within a for loop
b. Shift the Viewport and flip the *Window*

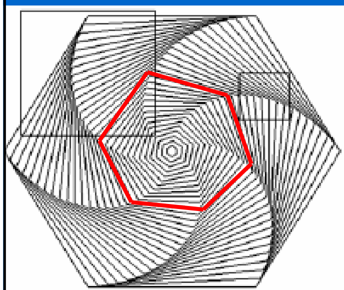
Modeling

How do we model the object on the left?



Modeling (2)

- **The model is a collection of concentric hexagons of various sizes, each rotated slightly with respect to the previous one.**

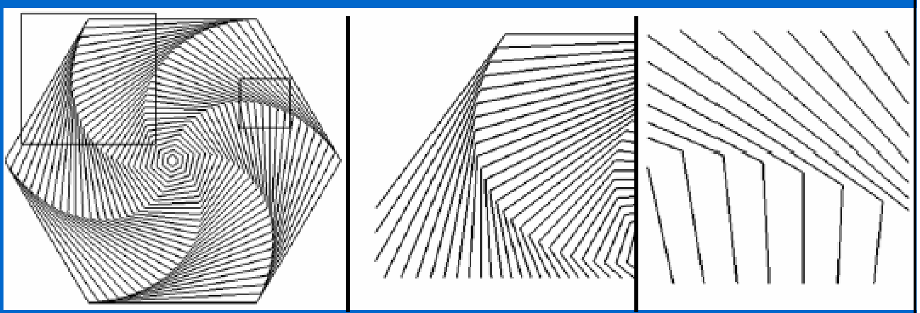


- Start with a small hexagon
  - (a list of vertices)
- Then scale and rotate in a loop

Clipping by the World Window

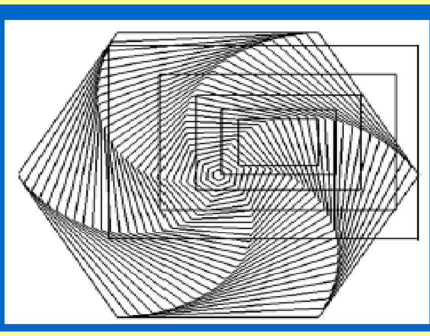
Clipping refers to viewing only the parts of an image that are in the World Window

Example: 2 Windows on the model



Applications: Zooming & Panning

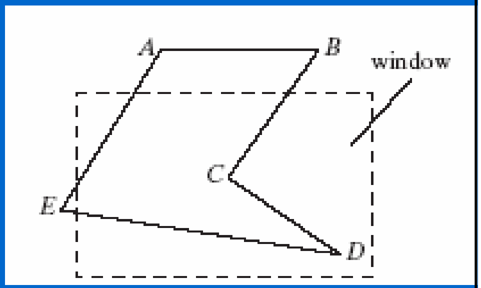
- *How to Zoom?*
- *Panning:* Moving camera over the world. *How?*



- **Zoom: concentric set of windows of decreasing size, displayed from outside in, into a viewport**
- **Pan: translating the window to a new position.**

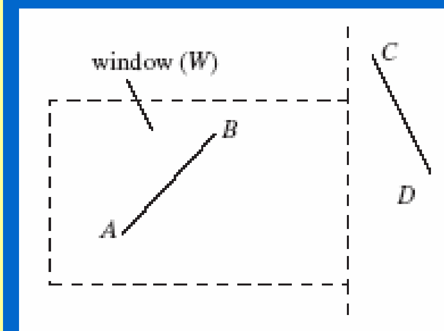
Clipping Line segments

- We want to draw only the parts of segment that are inside the World Window.
- chop portions outside the window
- *What are the simplest cases?*



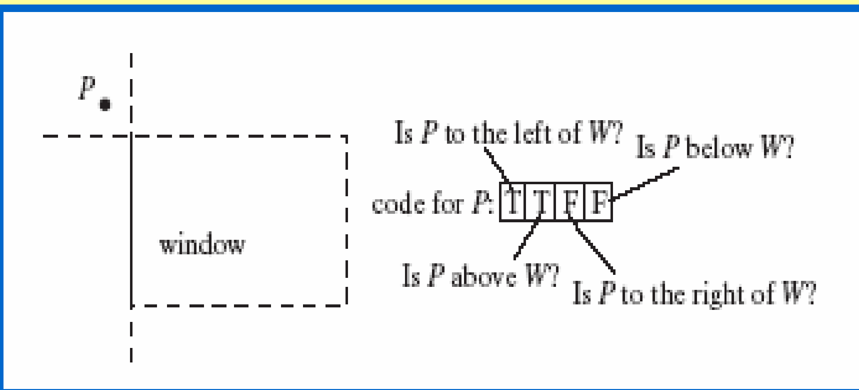
Cohen Sutherland Algorithm (1)

- 2 trivial cases:
- 1. segment AB totally inside the window
  - we draw all of
- Segment CD totally outside the window
  - we do not draw at all



Cohen Sutherland Algorithm (2)

- **we give each endpoint of a segment a 4 bit code specifying where it lies relative to the window W**
- ***How many states are possible?***



Cohen Sutherland Algorithm (3)

- **The diagram below shows Boolean codes for the 9 possible regions the endpoint lies in (left, above, below, right).**

TTFF	FTFF	FTTF
TFFF	FFFF window	FFTF
TFFT	FFFT	FFTT

Cohen Sutherland Algorithm (4)

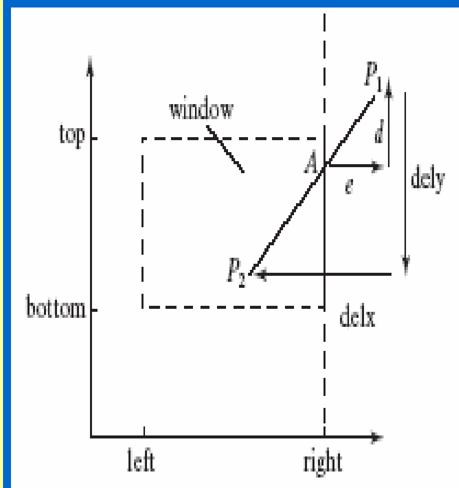
- **Do{ //* for a give segment P1P2**
  - **Form code-words for P1 and P2**
  - **If (trivial accept) return 1;**
  - **If (trivial reject) return 0;**
  - **Clip segment at next window border;**
    - **get new values for P1 and P2; discard outside part**

  } while (1)

- ***After how many iterations at most the algorithm stops?***
- **Result: endpoints of the clipped segments**

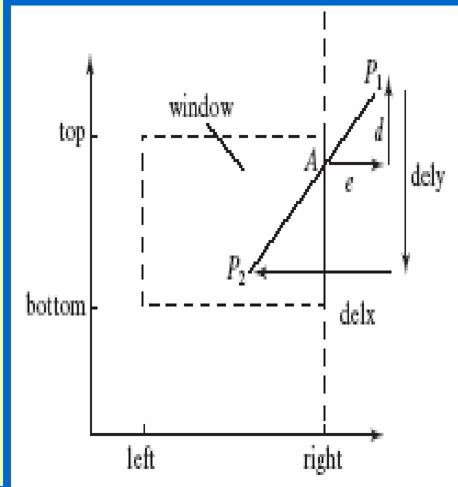
Clipping: replace (P_1x, P_1y) by (Ax, Ay)

- **We know the locations of P1 and P2**
- ***How?***
- **Assume P1 is to the right of *right* boundary**
- **We know Ax**
- ***How?***



Clipping: replace (P_1x, P_1y) by (Ax, Ay)

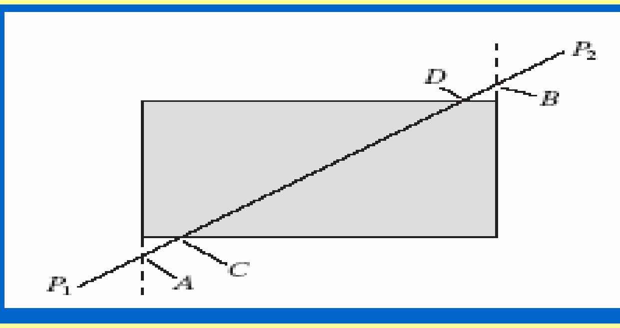
- $Ax = Wr$ (known)
- $Ay = ?$
- $d/dely = e/delx$
  - $d = e*(dely/delx)$
- $delx = P_1x - P_2x$
- $dely = P_1y - P_2y$
- $e = P_1x - Wr$.
- $Ay = P_1y - d$.



$$Ay = P_1y + (Wr-P_1x))(P_1y-P_2y)/(P_1x-P_2x)$$

Cohen Sutherland Algorithm (5)

- Change P1 to A, then P2 to B, then A to C, then P2 to D

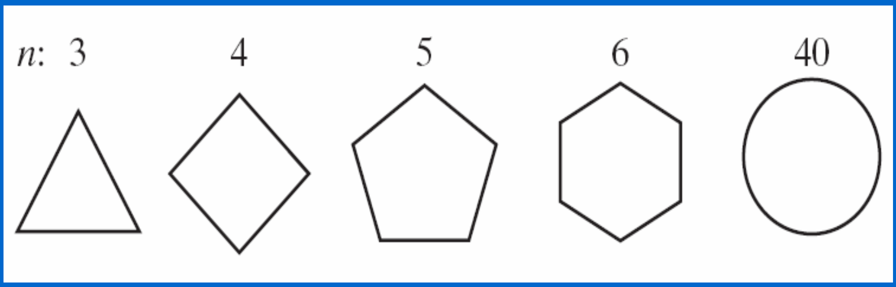


- Equation of line: = mx + b
- Point D: y = W.t, → x = (W.t – b)/m
- Point B: x = W.r → y = m*W.r + b

Drawing Regular Polygons, Circles, and Arcs

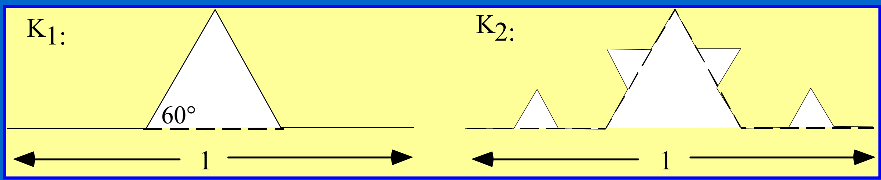
- **A polygon is <u>simple</u> if no two of its edges cross each other. More precisely, only adjacent edges can touch and only at their shared endpoint.**
- **A polygon is <u>regular</u> if it is simple, if all its sides have equal length, and if adjacent sides meet at equal interior angles.**

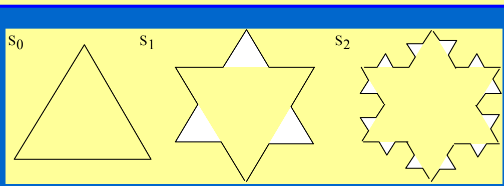
Regular Polygons



“Refinement” Transformation: Koch Curves

- Start with a line segment
- Each line segment is refined
  - – Replaced by a 4 segments bump



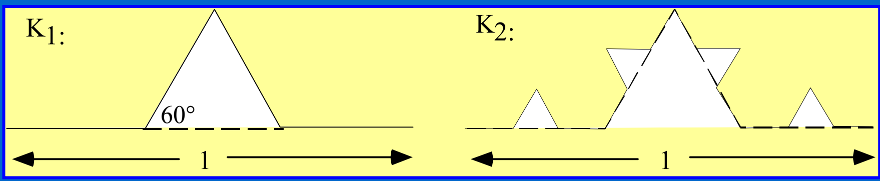


Koch Curves (2)

- Successive generations of the Koch curve are denoted K_0, K_1, K_2,…
- The 0-th generation shape K_0 is just a horizontal line of length 1.
- The curve K_1 is created by dividing the line K_0 into three equal parts, and replacing the middle section with a triangular bump having sides of length 1/ 3.
- The total line length is 4 / 3.

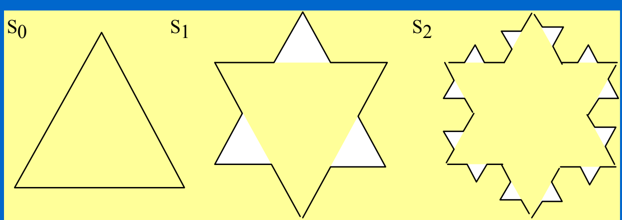
Koch Curves (2)

- The second-order curve K_2 is formed by building a bump on each of the four line segments of K_1.

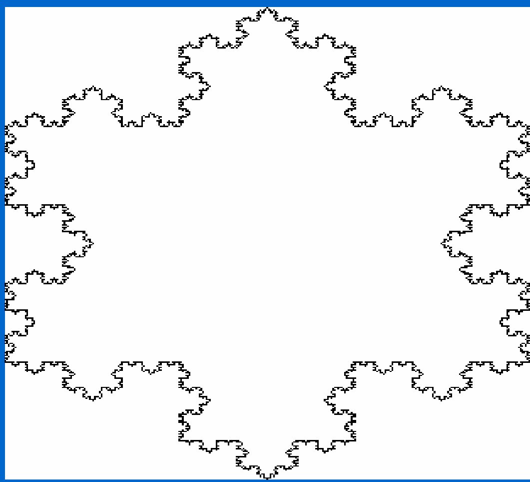


Koch Snowflake (3 joined curves)

- Perimeter: the *i*-th generation shape S_i is three times the length of a simple Koch curve, $3(4/3)^i$, which grows forever as *i* increases.
- Area inside the Koch snowflake: grows quite slowly, and in the limit, the area of S_∞ is only 8/5 the area of S_0.



Fifth-generation Koch Snowflake



How do we represent curves?

Representing Curves

- Three forms of equation for a given curve:
  - Explicit : e.g. line: $y = m*x + b$
  - Implicit: $F(x, y) = 0$; e.g., $y - m*x - b = 0$
  - Parametric: $x = x(t)$, $y = y(t)$, t a parameter;
  - frequently, $0 \leq t \leq 1$.
    - $\mathbf{P(t)} = \mathbf{P_1}*(1-t) + \mathbf{P_2}*t$. Line segment
    - $\mathbf{P_1}$ $\mathbf{P_2}$ and **P** are 2D points with x and y values.
    - $x = x_1*(1-t) + x_2*t$
    - $y = y_1*(1-t) + y_2*t$.

Specific Parametric Forms

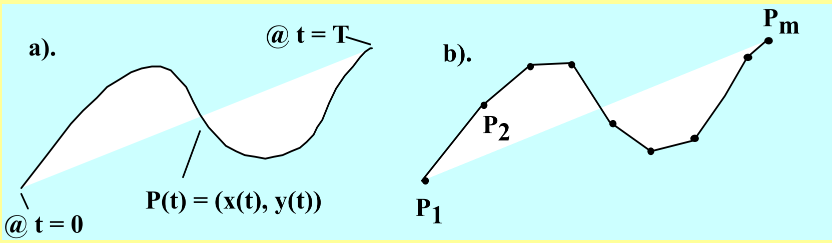
- line:
  - $x = x_1*(1-t) + x_2*t$; $y = y_1*(1-t) + y_2*t$
- circle: $x = r*\cos(2\pi\ t)$, $y = r*\sin(2\pi\ t)$
- ellipse: $x = W*r*\cos(2\pi\ t)$, $y = H*r*\sin(2\pi\ t)$
  - W and H are half-width and half-height
  - $0 <= t <= 1$
- *How do we find implicit form from parametric?*

Finding Implicit Form from Parametric Form

- Combine the x(t) and y(t) equations to eliminate t.
- ellipse: $x = W*r*\cos(2\pi\ t)$, $y = H*r*\sin(2\pi\ t)$
  - $X^2 = W^2r^2\cos^2(2\pi\ t)$, $y^2 = H^2r^2\sin^2(2\pi\ t)$.
  - Dividing by the W or H factors and adding gives $(x/W)^2 + (y/H)^2 = 1$, the implicit form.
- *How do we model a curve?*

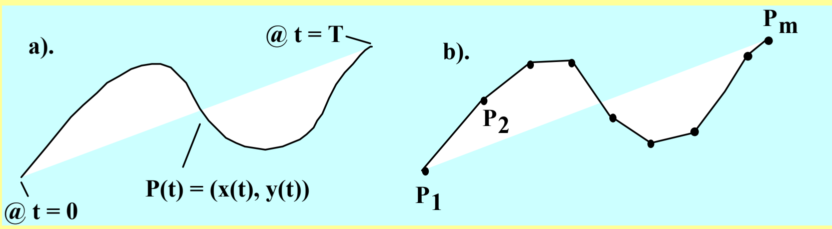
Modeling Curves

- **a curve *C* with the parametric form**
- **$P(t) = (x(t), y(t))$ as t varies from 0 to T**
- **we use samples of $P(t)$ at closely spaced instants.**



Modeling Curves (2)

- **The position $P_i = P(t_i) = (x(t_i), y(t_i))$ is calculated for a sequence $\{t_i\}$ of times.**
- **The curve $P(t)$ is approximated by the polyline based on this sequence of points P_i.**



Modeling Curves in OpenGL

- **Code:**

```
// draw the curve (x(t), y(t)) using
// the array t[0],..,t[n-1] of sample times
glBegin(GL_LINES);
   for(int i = 0; i < n; i++)
      glVertex2f((x(t[i]), y(t[i]));
glEnd();
```

Parametric Curves: Advantages

- **For Modeling/drawing purposes, parametric forms circumvent all of the difficulties of implicit and explicit forms.**
- **Curves can be multi-valued, and they can self-intersect any number of times.**
- **Verticality presents no special problem: $x(t)$ simply becomes constant over some interval in t.**