More Antialiasing, 2D Transformations, Matrices, and Homogeneous Coordinates

CS116A

Chris Pollett

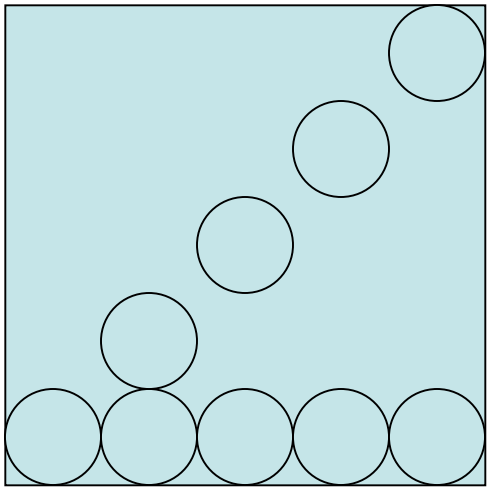
Oct 11, 2004.

Outline

- Line Intensity Issues
- Antialiasing Area Boundaries
- OpenGL Antialiasing Functions
- OpenGL Query Functions
- OpenGL Attribute Groups
- 2D Transformations
- Matrices and Homogeneous Coordinates
- Inverse Transformations

Line Intensity Issues

- Consider the two lines:

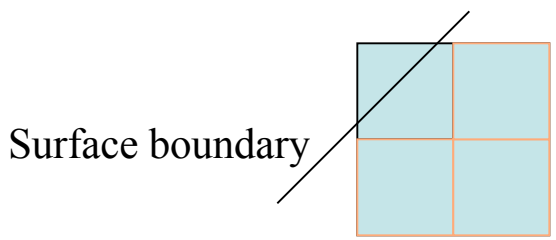


- Although they both have the same number of pixels, the second is (2)^{1/2} longer.
- If you use antialiasing this problem is compensated for automatically because the diagonal line will be drawn darker.

Antialiasing Area Boundaries

One can also smooth fill regions using antialiasing techniques:

- At the boundary of the object one adjusts the pixel intensity according to the fraction of the pixel that is interior to the object.



Pink subpixels are entirely within region-- intensity set according to the number of such subpixels

Pixel subdivided into 4 subpixels

Other Antialiasing Fill Techniques

Pitteway and Watkinson have a technique which is a variation on Bresenham Algorithm.

- Recall if |m| <1, we are choosing between (x_k+1, y_k) and (x_k+1, y_k+1) to be the next pixel to plot.
- If we add 1-m to the decision variable p we get a new decision variable p' equal to

  [m(x_k+1) +b] - (y_k +.5) + (1-m)
- One can now choose between points depending on if < 1-m or not.
- This p' also gives the area of the intersection of the current pixel with the line.
- Have to tweak this idea if have two lines intersect within a pixel

OpenGL AntiAliasing Functions

Antialiasing can be activated using:

```
glEnable(primitiveType);
```

where primitiveType is one of
GL_POINT_SMOOTH, GL_LINE_SMOOTH,
GL_POLYGON_SMOOTH.

To use this, one also needs to set up blending:

```
glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA,
GL_ONE_MINUS_SRC_ALPHA)
```

OpenGL Query Functions

Query functions are used to determine the current state of the OpenGL state machine.

One should use the appropriate OpenGL get function according to the type of the state variable one wants to find out about:

glGetBooleanv() glGetFloatv()

glGetIntegerv() glGetDoublev()

For example, to get an array with the current floating point color settings one could call:

```
glGetFloatv(GL_CURRENT_COLOR, colorValues);
```

Some useful flags are: GL_POINT_SIZE, GL_POINT_SIZE_RANGE, GL_LINE_WIDTH, GL_CURRENT_RASTER_POSITION,etc.

OpenGL Attribute Groups

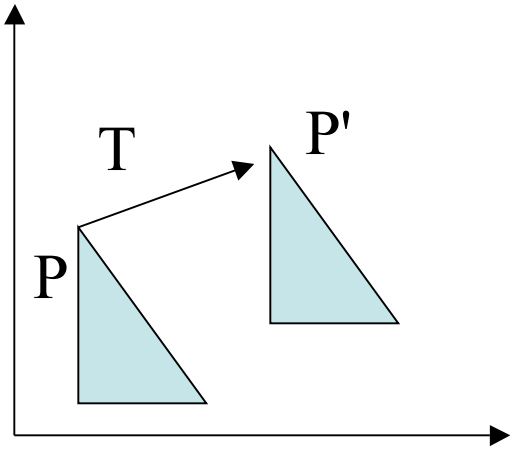
- OpenGL state parameters are arranged into groups called **attribute groups**. For example, one has the point-attribute group, the line attribute group, and the polygon attribute group.
- One can use the OpenGL server attribute stack to push and pop such groups of settings.
- For example, glPushAttrib(GL_POINT_BIT | GL_LINE_BIT | GL_POLYGON_BIT);
- To retrieve use: glPopAttrib().

2D Transformations

Given a 2D figure. One might want to:

- translate it
- rotate it
- scale it

Translations



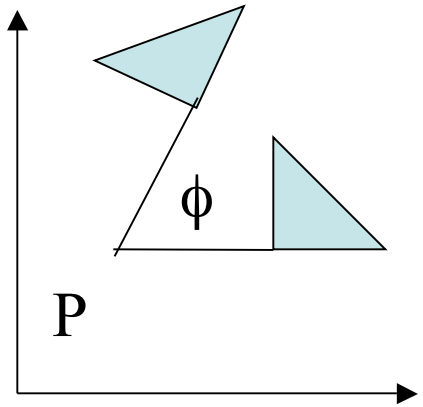
If the coordinates for P are (x,y) and for the translation T are (t_x, t_y). The new point will be: (x+t_x, y + t_y). Written using the matrix:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t\_x \\ t\_y \end{bmatrix}$$

Rotations

To do a rotation need to specify:

- A base point for the rotation
- An angle to rotate



The matrix for a rotation about the origin looks like:

$$\begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

Scaling

- Scalings are used to stretch the figure in either the x or y direction.
- For example a point (x,y) could be stretch to be (x' , y') via the maps x' = s_x*x and y' = s_y*y.
- The matrix for a scaling looks like:

$$\begin{bmatrix} s\_x & - & 0 \\ 0 & & s\_y \end{bmatrix}$$

Matrices and Homogeneous Coordinates

- So far we could represent the effect of the operations we have on a point with an equation like:

  newP = M P + T

  where M is a 2 x 2 matrix and everything else is a 2D column vector.
- It is more convenient if we want to do sequences of operations to have all our operations done using matrix multiplication
- To get this to work we need to use homogeneous coordinates for points in 2D. A point (x,y) is mapped to the point (x, y, 1) in homogeneous coordinates.
- We say to 3D vectors v1=(x1,y1,z1) and v2=(x2,y2,z2) represent the same 2D point if there is an h such that (x1, y1, z1) = (h*x2, h*y2, h*z2).

Matrices in Homogeneous Coordinates

Translation:

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 1 & 1 \end{bmatrix}$$

Rotation:

$$\begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Scaling:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Transformations

- Each of the operations discussed today can be undone.
- For example, the inverse of the translation T of the last slide, T^{-1}, can be undone using the matrix:

$$\begin{bmatrix} 1 & 0 & -t\_x \\ 0 & 1 & -t\_y \\ 0 & 1 & 1 \end{bmatrix}$$