
CIS 454 Computer Graphics

Lecture 10, 10/05/2006

Li Shen
Computer and Information Science
UMass Dartmouth

Better Interactive Programs

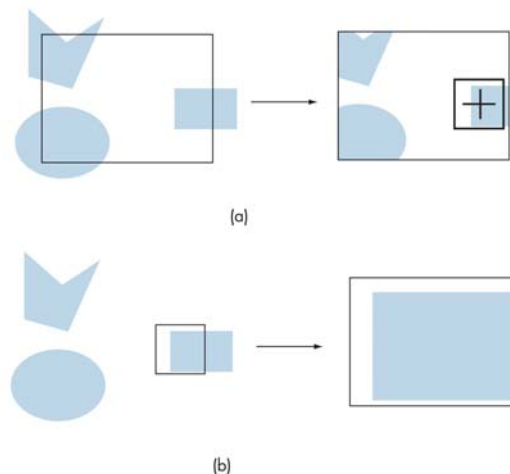
- Learn to build more sophisticated interactive programs using
 - Picking
 - Select objects from the display
 - Three methods
 - Rubberbanding
 - Interactive drawing of lines and rectangles
 - Display Lists
 - Retained mode graphics
-

Three Approaches for Picking

- Hit list
 - Clipping region: most general approach but most difficult to implement
 - Use back or some other buffer to store object ids as the objects are rendered
 - Rectangular maps
 - Easy to implement for many applications
 - See [paint](#) program in text
-

Hit List

- Adjust clipping region
- Keep tracking of which primitives are rendered
- Include those into a hit list

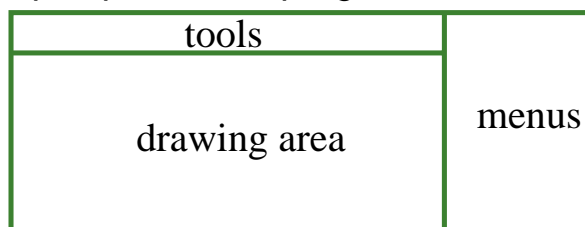


Using Another Buffer and Colors

- For a small number of objects, we can assign a unique color (often in color index mode) to each object
 - We then render the scene to a color buffer other than the front buffer so the results of the rendering are not visible
 - We then get the mouse position and use `glReadPixels()` to read the color in the buffer we just wrote at the position of the mouse
 - The returned color gives the id of the object
-

Using Regions of the Screen

- Many applications use a simple rectangular arrangement of the screen
 - Example: paint/CAD program

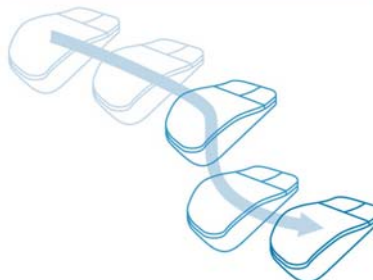
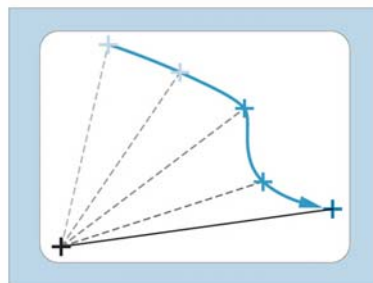


- Easier to look at mouse position and determine which area of screen it is in than using selection mode picking
-

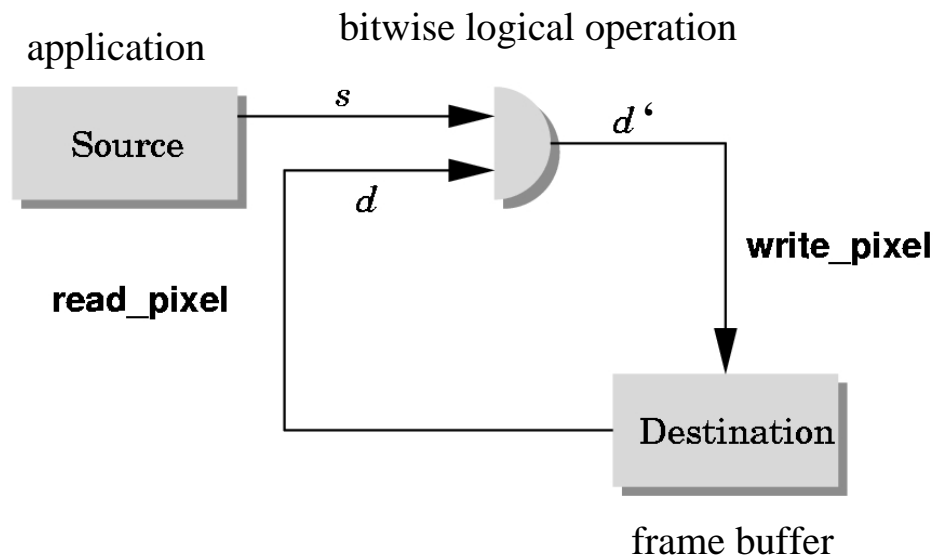
Outline

- Learn to build more sophisticated interactive programs using
 - Picking
 - Select objects from the display
 - Three methods
 - Rubberbanding
 - Interactive drawing of lines and rectangles
 - Display Lists
 - Retained mode graphics
-

Rubberbanding



Writing Modes



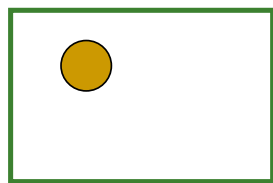
XOR write

- Usual (default) mode: source replaces destination ($d' = s$)
 - Cannot write temporary lines this way because we cannot recover what was “under” the line in a fast simple way
- Exclusive OR mode (XOR) ($d' = d \oplus s$)
 - $x \oplus y \oplus x = y$
 - Hence, if we use XOR mode to write a line, we can draw it a second time and line is erased!

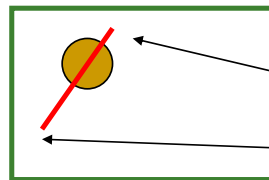
Rubberbanding

- Switch to XOR write mode
- Draw object
 - For line can use first mouse click to fix one endpoint and then use motion callback to continuously update the second endpoint
 - Each time mouse is moved, redraw line which erases it and then draw line from fixed first position to to new second position
 - At end, switch back to normal drawing mode and draw line
 - Works for other objects: rectangles, circles

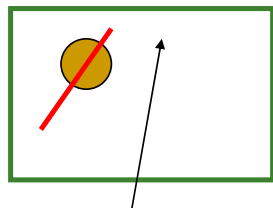
Rubberband Lines



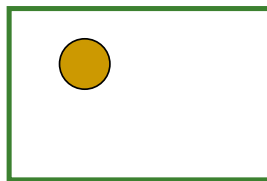
initial display



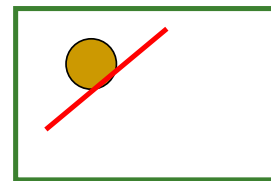
draw line with mouse
in XOR mode



mouse moved to
new position



original line redrawn
with XOR



new line drawn
with XOR

XOR in OpenGL

- There are **16** possible logical operations between two bits
 - All are supported by OpenGL
 - Must first enable logical operations
 - `glEnable(GL_COLOR_LOGIC_OP)`
 - Choose logical operation
 - `glLogicOp(GL_XOR)`
 - `glLogicOp(GL_COPY)` (default)
 - See textbook pages 148-150 for sample code
-

Outline

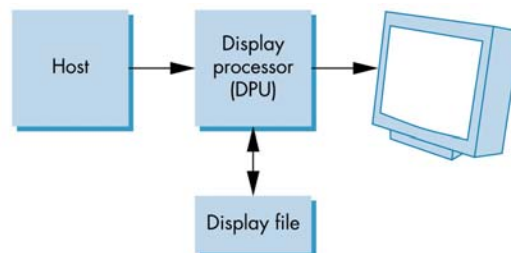
- Learn to build more sophisticated interactive programs using
 - Picking
 - Select objects from the display
 - Three methods
 - Rubberbanding
 - Interactive drawing of lines and rectangles
 - Display Lists
 - Retained mode graphics
-

Immediate and Retained Modes

- Recall that in a standard OpenGL program, once an object is rendered there is no memory of it and to redisplay it, we must re-execute the code for it
 - Known as *immediate mode graphics*
 - Can be especially slow if the objects are complex and must be sent over a network
- Alternative is define objects and keep them in some form that can be redisplayed easily
 - *Retained mode graphics*
 - Accomplished in OpenGL via *display lists*

Display Lists

- Conceptually similar to a graphics file
 - Must define (name, create)
 - Add contents
 - Close
- In client-server environment, display list is placed on server
 - Can be redisplayed without sending primitives over network each time



Display List Functions

- Creating a display list

```
GLuint id;  
  
void init()  
{  
    id = glGenLists( 1 );  
    glNewList( id, GL_COMPILE );  
    /* other OpenGL routines */  
    glEndList();  
}
```

- Call a created list

```
void display()  
{  
    glCallList( id );  
}
```

- GL_COMPILE

- Send the list to the server
- Not to display

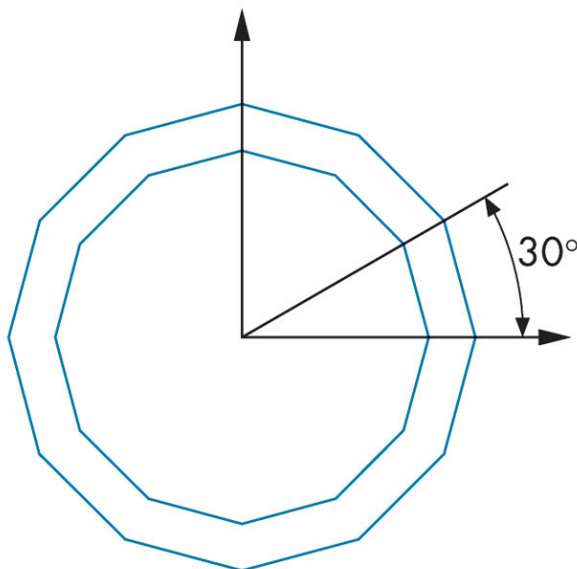
- GL_COMPILE_AND_EXECUTE

- Immediate display

- Redisplay

- glCallList

Drawing of the Letter “O”



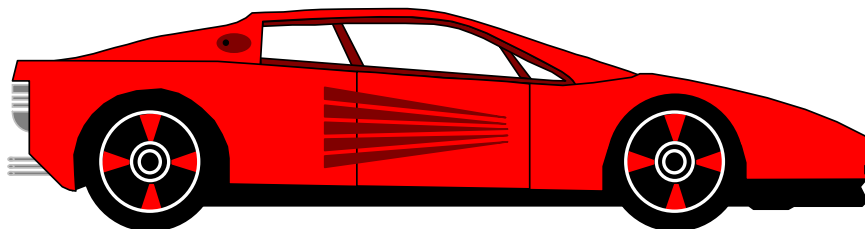
Display Lists and State

- Most OpenGL functions can be put in display lists
- State changes made inside a display list persist after the display list is executed
- Can avoid unexpected results by using `glPushAttrib` and `glPushMatrix` upon entering a display list and `glPopAttrib` and `glPopMatrix` before exiting

Hierarchy and Display Lists

- Consider model of a car
 - Create display list for chassis
 - Create display list for wheel

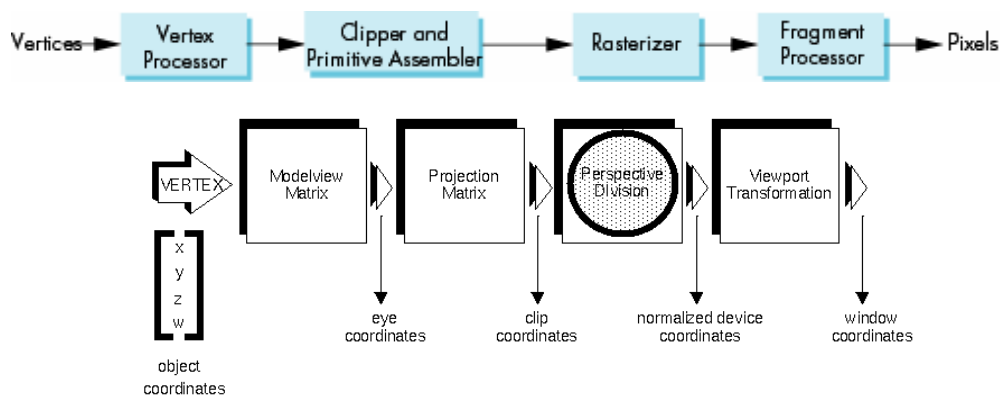
```
glNewList( CAR, GL_COMPILE );  
  glCallList( CHASSIS );  
  glTranslatef( ... );  
  glCallList( WHEEL );  
  glTranslatef( ... );  
  glCallList( WHEEL );  
  ...  
glEndList();
```



Summary: Input and Interaction

- Introduction
 - Input devices: physical vs logical
 - Input Modes: Event-driven input
 - Smooth animations: double buffering
 - Programming event input with GLUT
- Working with Callbacks
 - Mouse, keyboard, reshape
 - Menus
- Better interactive programs
 - Picking: select objects from the display
 - Rubberbanding: interactive drawing of lines and rectangles
 - Display Lists: retained mode graphics

Next Topic: Geometric Objects and Transformations



- Geometry: scalars, points, vectors
- Representation: coordinate systems and frames
- Transformation: rotation, translation, scaling, shear
- OpenGL transformation: `matrix_mode`
- Building models: polygonal models in OpenGL

[gasket3](#)

Geometry

Objectives

- Introduce the elements of geometry
 - Scalars
 - Vectors
 - Points
 - Develop mathematical operations among them in a coordinate-free manner
 - Define basic primitives
 - Line segments
 - Polygons
-

Basic Elements

- Geometry is the study of the relationships among objects in an n-dimensional space
 - In computer graphics, we are interested in objects that exist in three dimensions
 - Want a minimum set of primitives from which we can build more sophisticated objects
 - We will need three basic elements
 - Scalars
 - Vectors
 - Points
-

Coordinate-Free Geometry

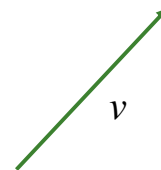
- When we learned simple geometry, most of us started with a Cartesian approach
 - Points were at locations in space $\mathbf{p}=(x,y,z)$
 - We derived results by algebraic manipulations involving these coordinates
 - This approach was nonphysical
 - Physically, points exist regardless of the location of an arbitrary coordinate system
 - Most geometric results are independent of the coordinate system
 - Example Euclidean geometry: two triangles are identical if two corresponding sides and the angle between them are identical
-

Scalars

- Need three basic elements in geometry
 - Scalars, Vectors, Points
- Scalars can be defined as members of sets which can be combined by two operations (addition and multiplication) obeying some fundamental axioms (associativity, commutativity, distributivity)
- Examples include the real and complex number systems under the ordinary rules with which we are familiar
- Scalars alone have no geometric properties

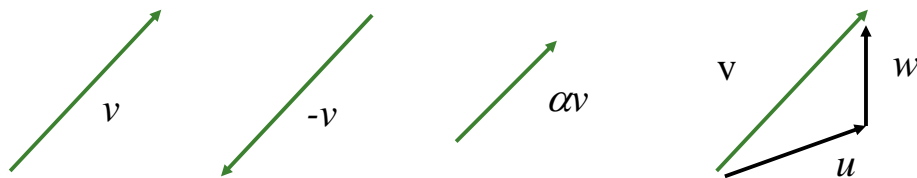
Vectors

- Physical definition: a vector is a quantity with two attributes
 - Direction
 - Magnitude
- Examples include
 - Force
 - Velocity
 - Directed line segments
 - Most important example for graphics
 - Can map to other types



Vector Operations

- Every vector has an inverse
 - Same magnitude but points in opposite direction
- Every vector can be multiplied by a scalar
- There is a zero vector
 - Zero magnitude, undefined orientation
- The sum of any two vectors is a vector
 - Use head-to-tail axiom

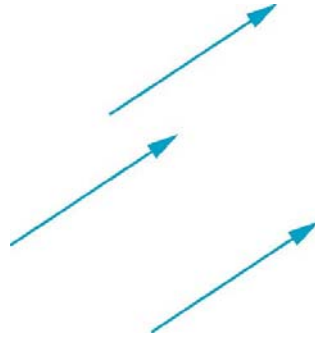


Linear Vector Spaces

- Mathematical system for manipulating vectors
- Operations
 - Scalar-vector multiplication $u = \alpha v$
 - Vector-vector addition: $w = u + v$
- Expressions such as
$$v = u + 2w - 3r$$
Make sense in a vector space

Vectors Lack Position

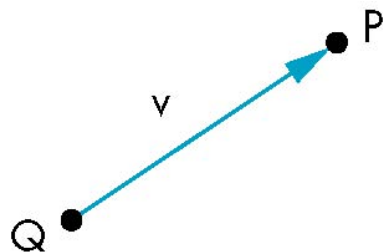
- These vectors are identical
 - Same length and magnitude



- Vectors spaces insufficient for geometry
 - Need points

Points

- Location in space
- Operations allowed between points and vectors
 - Point-point subtraction yields a vector
 - Equivalent to point-vector addition



$$v = P - Q$$

$$P = v + Q$$

Affine Spaces

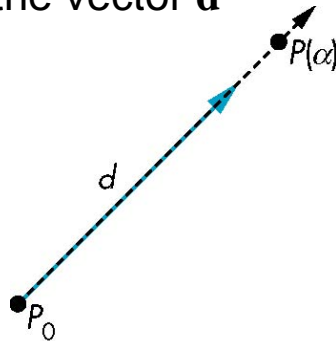
- Point + a vector space
 - Operations
 - Vector-vector addition
 - Scalar-vector multiplication
 - Point-vector addition
 - Scalar-scalar operations
 - For any point define
 - $1 \cdot P = P$
 - $0 \cdot P = \mathbf{0}$ (zero vector)
-

Objectives

- Introduce the elements of geometry
 - Scalars
 - Vectors
 - Points
 - Develop mathematical operations among them in a coordinate-free manner
 - Define basic primitives
 - Line segments
 - Polygons
-

Lines

- Consider all points of the form
 - $P(\alpha) = P_0 + \alpha \mathbf{d}$
 - Set of all points that pass through P_0 in the direction of the vector \mathbf{d}



Parametric Form

- This form is known as the parametric form of the line
 - More robust and general than other forms
 - Extends to curves and surfaces
- Two-dimensional forms
 - Explicit: $y = mx + h$
 - Implicit: $ax + by + c = 0$
 - Parametric:
 - $x(\alpha) = \alpha x_0 + (1-\alpha)x_1$
 - $y(\alpha) = \alpha y_0 + (1-\alpha)y_1$

Rays and Line Segments

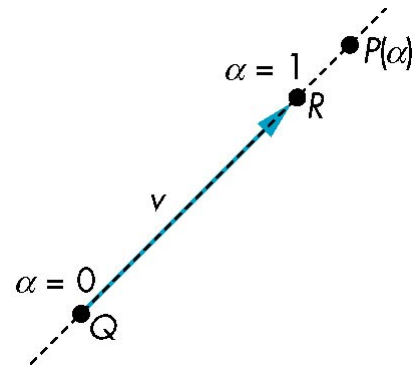
- If $\alpha \geq 0$, then $P(\alpha)$ is the *ray* leaving P_0 in the direction \mathbf{d}

If we use two points to define \mathbf{v} , then

$$P(\alpha) = Q + \alpha (R - Q) = Q + \alpha \mathbf{v}$$

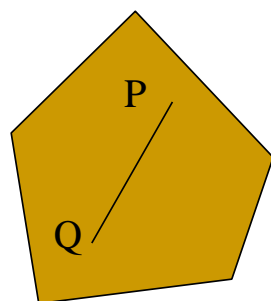
$$= \alpha R + (1 - \alpha)Q$$

For $0 \leq \alpha \leq 1$ we get all the points on the *line segment* joining R and Q

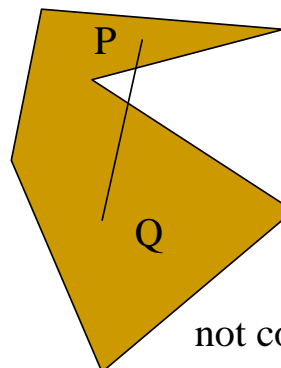


Convexity

- An object is *convex* iff for any two points in the object all points on the line segment between these points are also in the object



convex



not convex

Affine Sums

- Consider the “sum”

$$P = \alpha_1 P_1 + \alpha_2 P_2 + \dots + \alpha_n P_n$$

Can show by induction that this sum makes sense iff

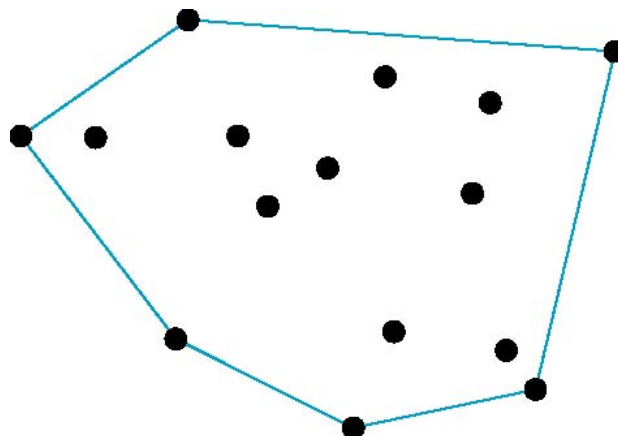
$$\alpha_1 + \alpha_2 + \dots + \alpha_n = 1$$

in which case we have the *affine sum* of the points P_1, P_2, \dots, P_n

- If, in addition, $\alpha_i \geq 0$, we have the *convex hull* of P_1, P_2, \dots, P_n

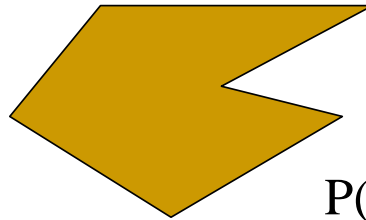
Convex Hull

- Smallest convex object containing P_1, P_2, \dots, P_n
- Formed by “shrink wrapping” points



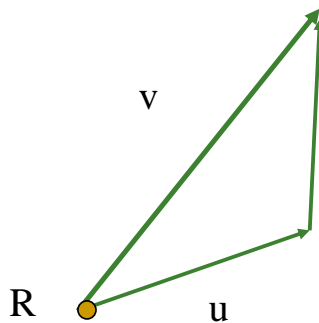
Curves and Surfaces

- Curves are one parameter entities of the form $P(\alpha)$ where the function is nonlinear
- Surfaces are formed from two-parameter functions $P(\alpha, \beta)$
 - Linear functions give planes and polygons

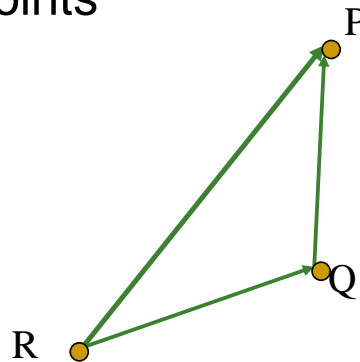

 $P(\alpha)$

 $P(\alpha, \beta)$

Planes

- A plane can be defined by a point and two vectors or by three points



$$P(\alpha, \beta) = R + \alpha u + \beta v$$



$$P(\alpha, \beta) = R + \alpha(Q - R) + \beta(P - Q)$$

After Class

- Read
 - Chapters 3,4
 - Appendices B,C
- Work on [HW2](#)

