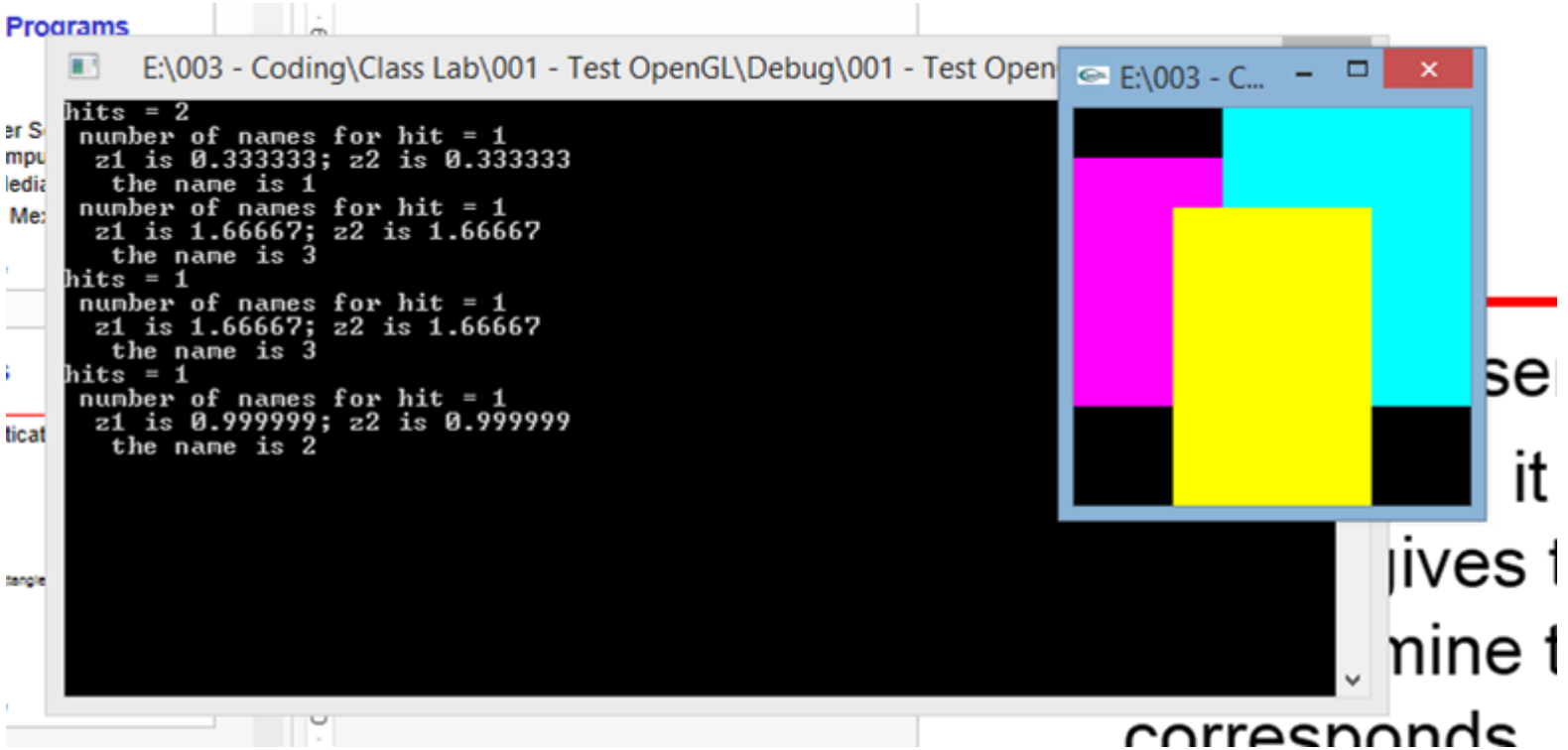
You must read this slide together with the text book or Angel Powerpoint slides

Better Interactive Programs

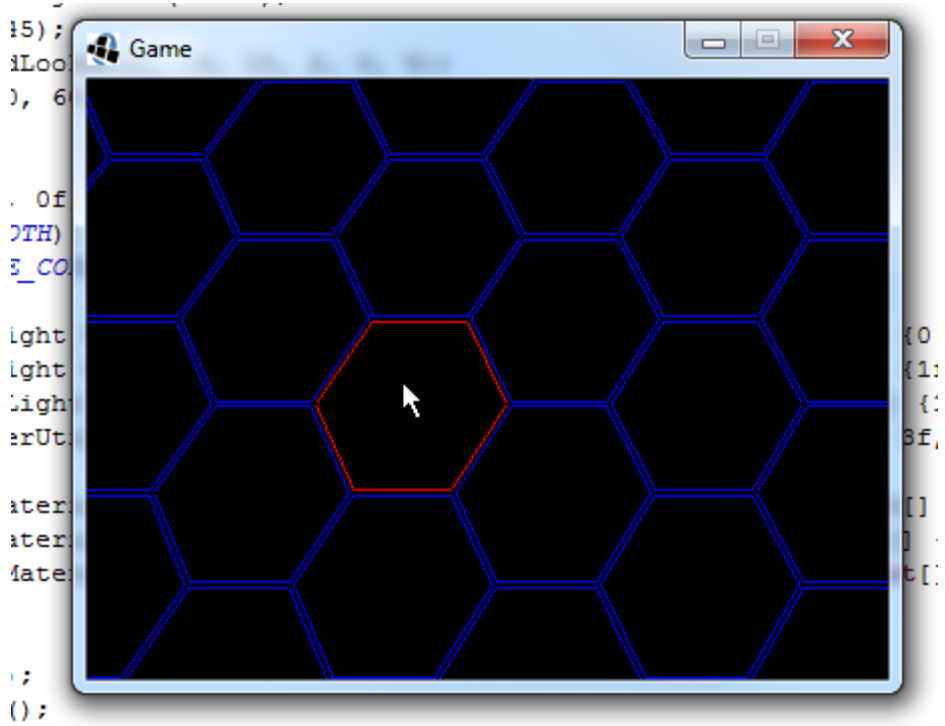
Hamzah Asyrani Sulaiman

Picking in OpenGL environment

In principle, it should be simple because the mouse gives the position and we should be able to determine to which object(s) a position corresponds



https://www.opengl.org/archives/resources/code/samples/redbook/pickdepth.c



Picking in OpenGL environment

http://schabby.de/picking-opengl-ray-tracing/

Practical difficulties

- Pipeline architecture is feed forward, hard to go from screen back to world
- Complicated by screen being 2D, world is 3D
- How close do we have to come to object to say we selected it?

Picking in OpenGL environment

1. Hit list
   - Most general approach but most difficult to implement
2. Use back or some other buffer to store object ids as the objects are rendered
3. Rectangular maps
   - Easy to implement for many applications
   - See paint program in text

Rendering Modes

OpenGL can render in one of three modes selected by **`glRenderMode(mode)`**

- **`GL_RENDER`**: normal rendering to the frame buffer (default)
- **`GL_FEEDBACK`**: provides list of primitives rendered but no output to the frame buffer
- **`GL_SELECTION`**: Each primitive in the view volume generates a *hit record* that is placed in a *name stack* which can be examined later

http://medialab.di.unipi.it/web/IUM/Waterloo/node87.html

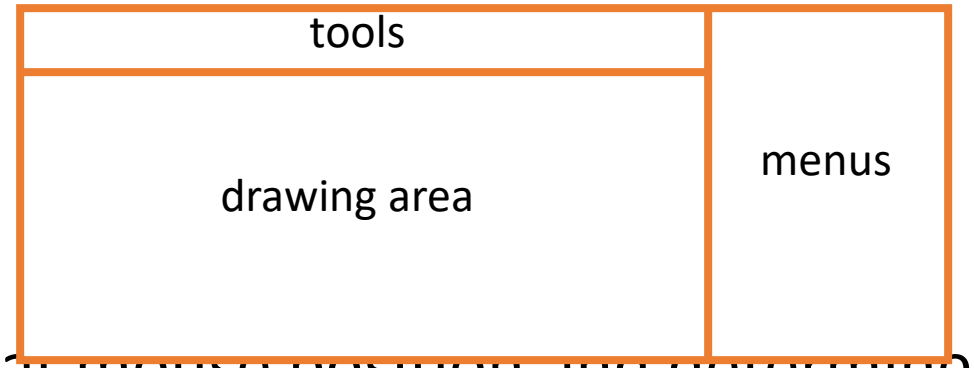
Selection Mode Functions

- **`glSelectBuffer()`**: specifies name buffer
- **`glInitNames()`**: initializes name buffer
- **`glPushName(id)`**: push id on name buffer
- **`glPopName()`**: pop top of name buffer
- **`glLoadName(id)`**: replace top name on buffer

id is set by application program to identify objects

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

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 );
}
```

See Examples

http://www.videotutorialsrock.com/opengl_tutorial/display_lists/text.php

Modified Version – Do it yourself

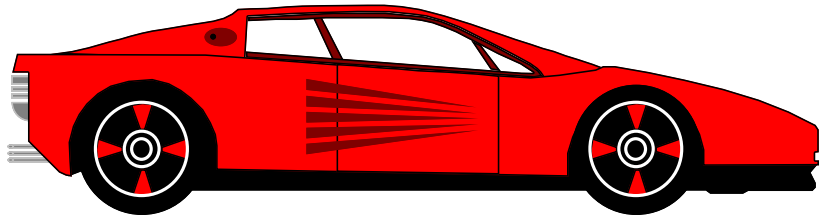
Display Lists and State

- 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();
```



```
// OBJECTS CONFIG_____________________________________
void CreateList(cObjConf *obj, int i, GLuint *dlist)
{
*dlist = glGenLists (i);
glNewList(*dlist, GL_COMPILE);
gLobal->DrawObj(obj);
glEndList();
};

void ListofAABB (cObjConf *obj,cAABB *aabb, int j,GLuint *dlisttree)
{
*dlisttree = glGenLists (j);
glNewList(*dlisttree, GL_COMPILE);
gAABB->getdata(aabb,obj);
glEndList();
};
```

CreateList(&Object3D[1],2,&List[1]);
ListofAABB(&Object3D[1],&AABBClss[1],1,&ListAABB[1]);

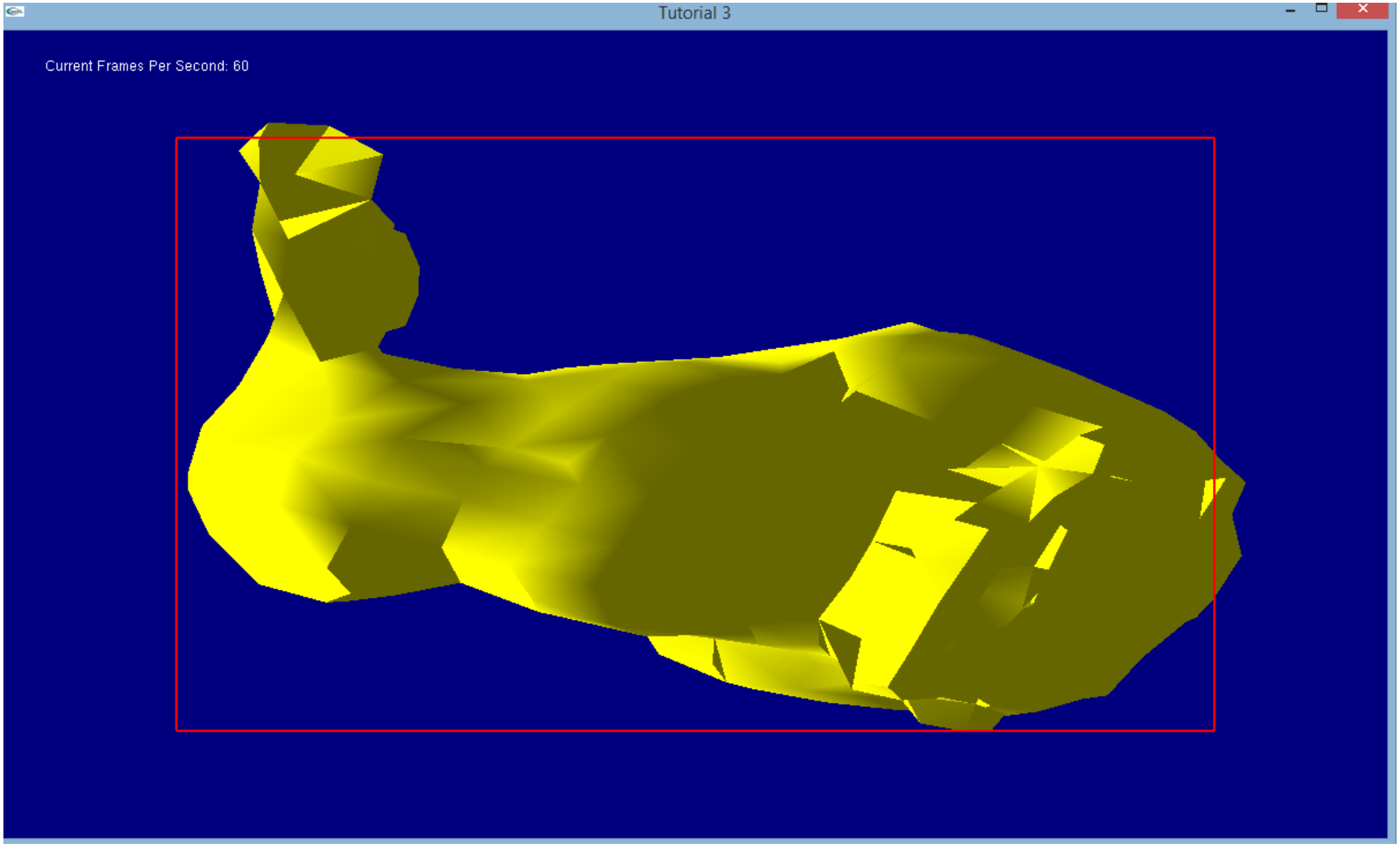
```
void DrawObject (void) {

glPushMatrix();
glPushAttrib(GL_ALL_ATTRIB_BITS);
//glTranslatef(tranX+0.9,tranY-1.0,tranZ);
//glTranslatef(angle-1.0,1.0,0);
glRotatef(-spin, 0,1,0);
glRotatef(spin1, 1,0,0);
//glScalef(0.4,0.4,0.4);
glGetFloatv(GL_MODELVIEW_MATRIX, Mat[1]);
glEnable(GL_COLOR_MATERIAL);
glEnable(GL_LIGHTING);
glColor3f(2.0,2.0,0.0);
glCallList(List[1]);
glEnable(GL_COLOR_MATERIAL);
glDisable(GL_LIGHTING);
//glLoadIdentity();
glDisable(GL_LIGHTING);
glDisable(GL_CULL_FACE);
glColor3f (1.0,0.0,0.0);
glPolygonMode (GL_FRONT_AND_BACK,GL_LINE);


gAABB->UpdateAABB(&AABBClss[1],Mat[1],&Object3D[1]);
gAABB->AABBBound(&AABBClss[1]);

glPopMatrix();
glPopAttrib();

}
```

```
glPushMatrix();
glPushAttrib(GL_ALL_ATTRIB_BITS);
```

```
glCallList(List[1]);
```

```
gAABB-
>UpdateAABB(&AABBClss[1],Mat[1],&Object3D[1]);
```

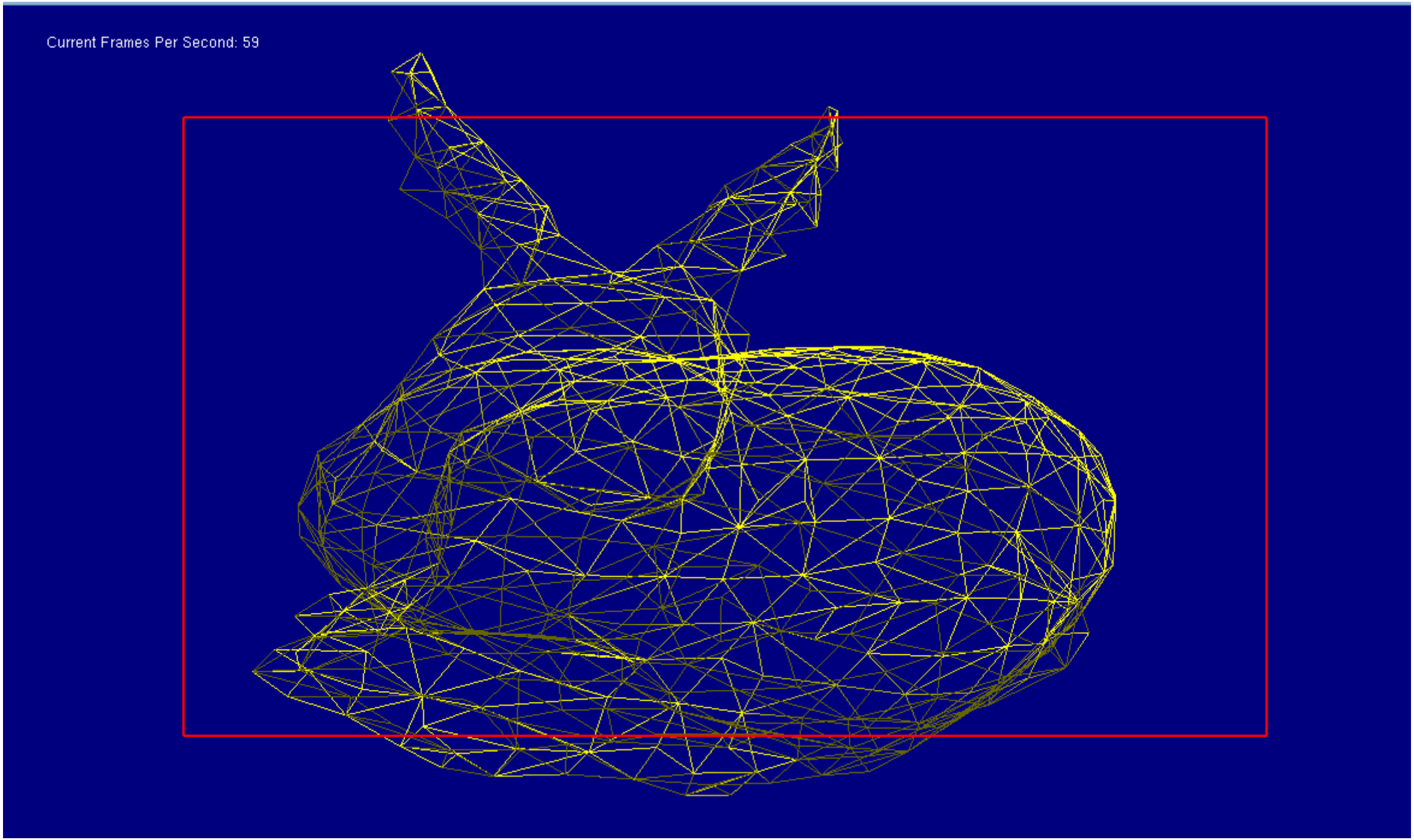
```
glPopMatrix();
glPopAttrib();
```


Tutorial 3
Current Frames Per Second: 60

```
glPushMatrix();
glPushAttrib(GL_ALL_ATTRIB_BITS);
```

```
glPopMatrix();
glPopAttrib();
```

Is removed


Current Frames Per Second: 59

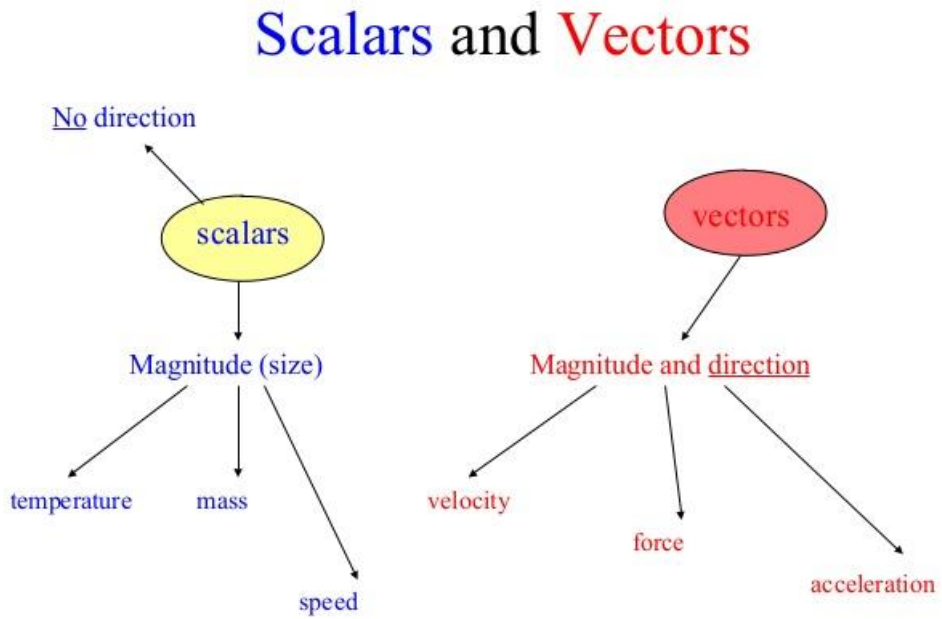
You must read this slide together with the text book or Angel Powerpoint slides

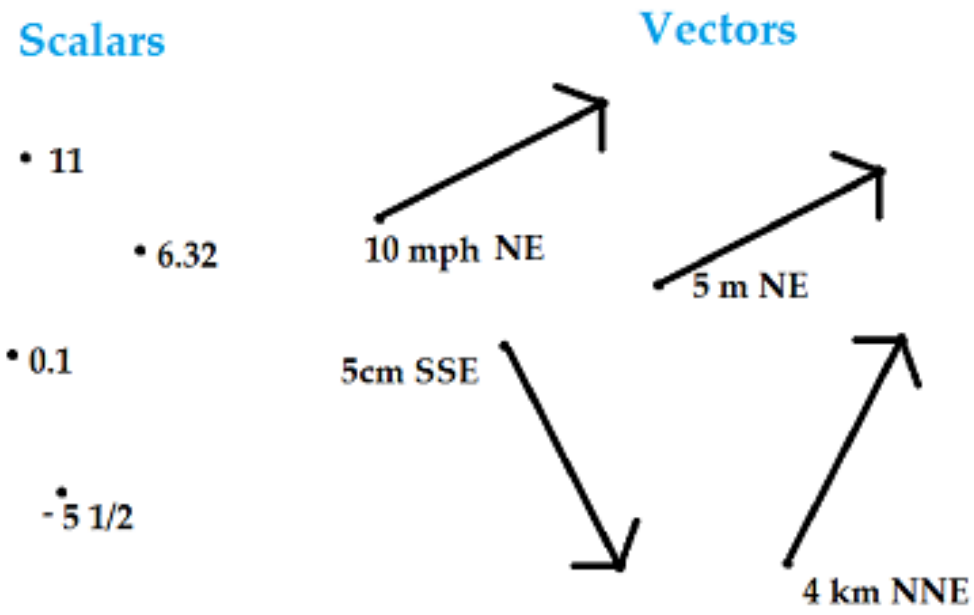
Geometry

Hamzah Asyrani Sulaiman

Scalars
Vectors
Points

three basic elements



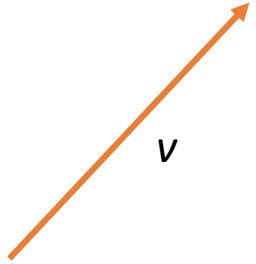


Scalars

- 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, commutivity, inverses)
- Examples include the real and complex number systems under the ordinary rules with which we are familiar
- Scalars alone have no geometric properties

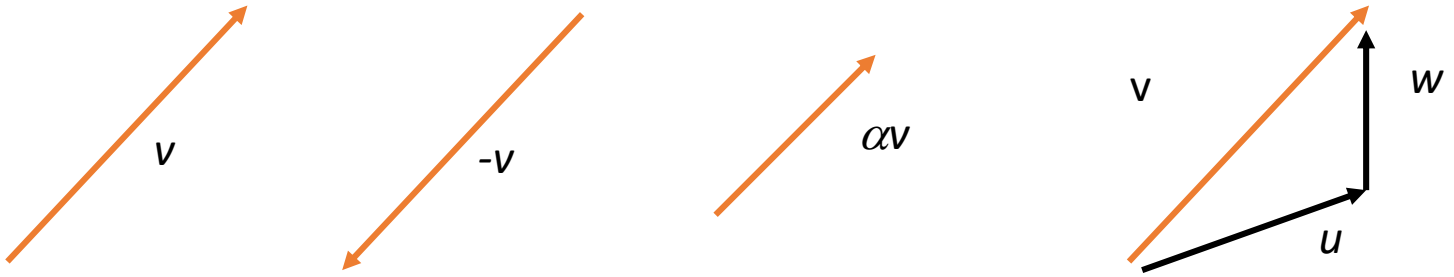
Vectors

- 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

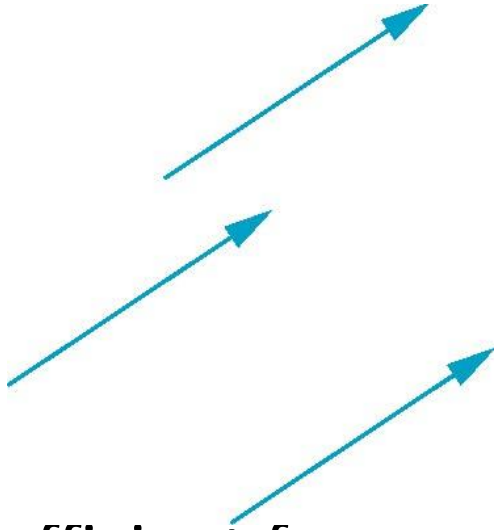
- Mathematical system for manipulating vectors
- Operations
  - Scalar-vector multiplication $u=\alpha v$
  - Vector-vector addition: $w=u+v$
- Expressions such as

  $v=u+2w-3r$

Make sense in a vector space

Vectors Lack Position

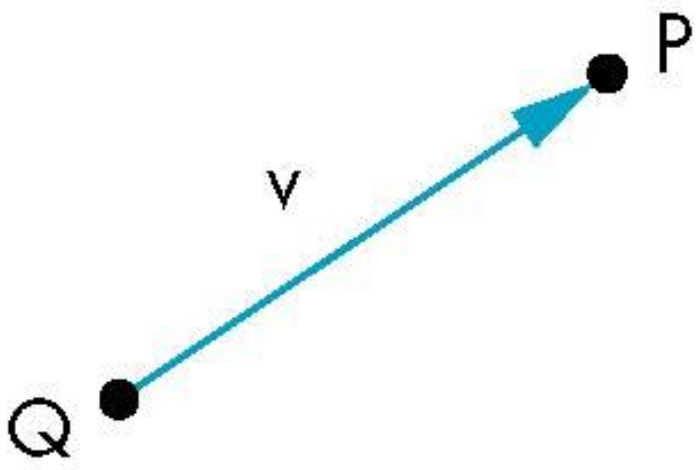
- These vectors are identical
  - Same length and magnitude



- 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

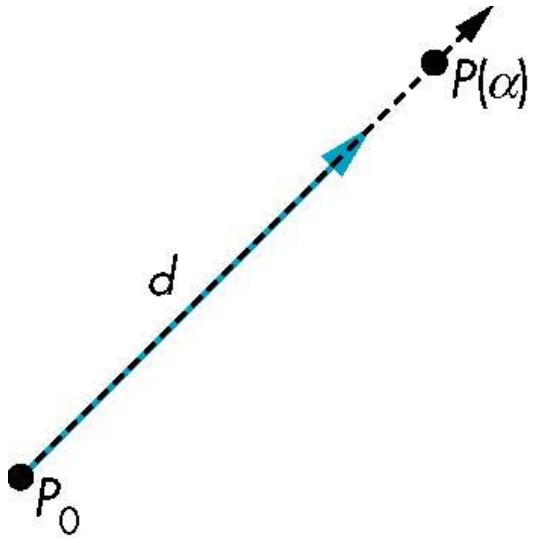
P=v+Q

Affine Spaces

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

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

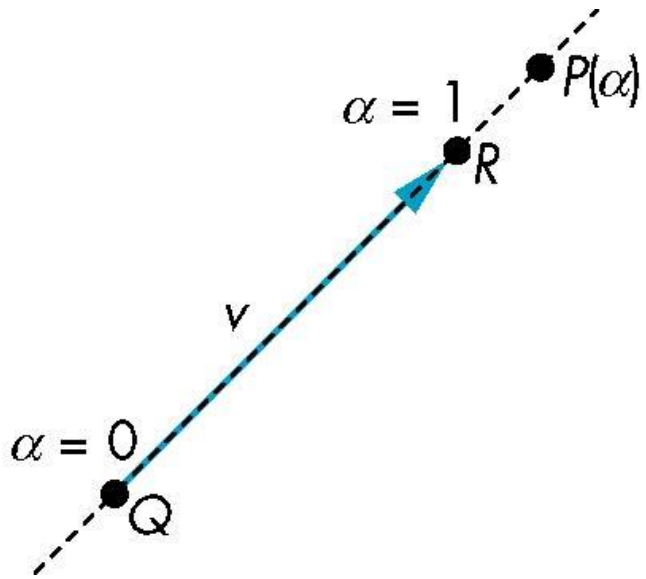
- If $\alpha >= 0$, then $P(\alpha)$ is the *ray* leaving P_0 in the direction $\mathbf{d}$

If we use two points to define v, then

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

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

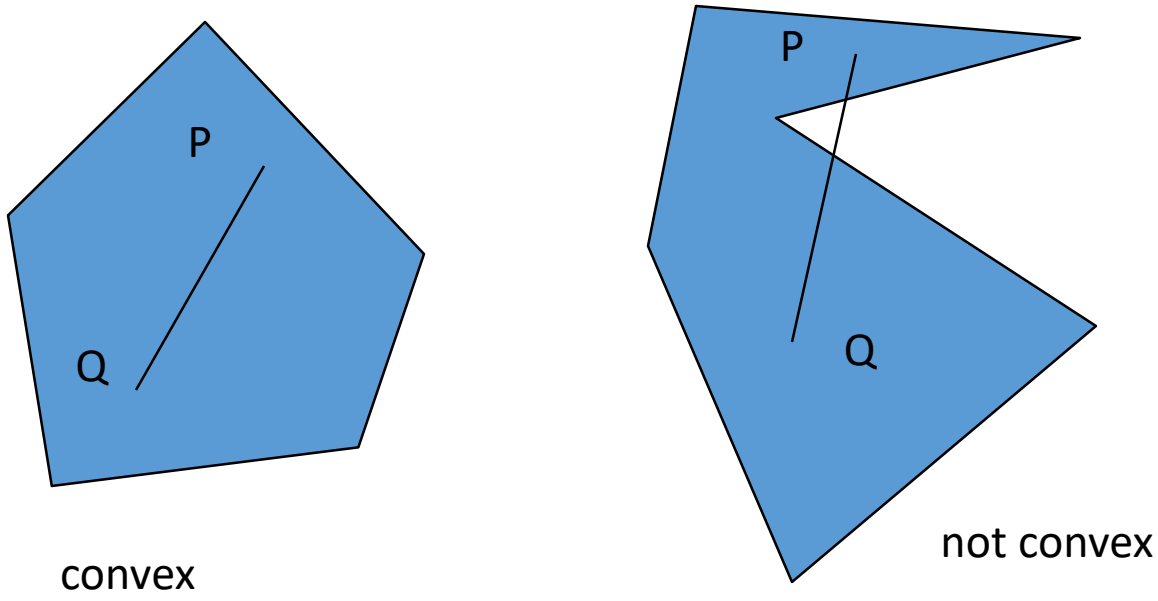
For $0 <= \alpha <= 1$ we get all the points on the *line segment* joining R and Q

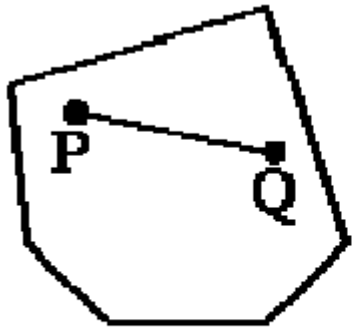


http://www.differencebetween.info/difference-between-convex-and-non-convex

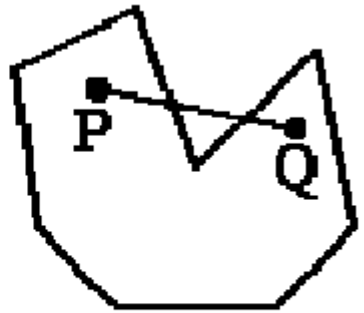
Convexity

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

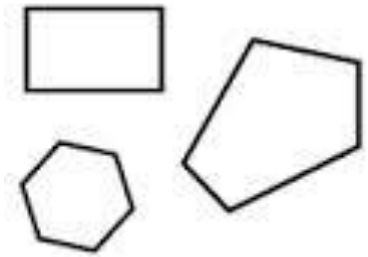




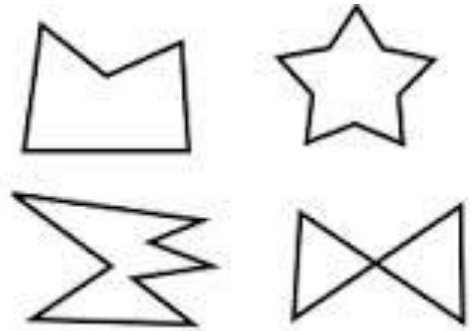
Convex



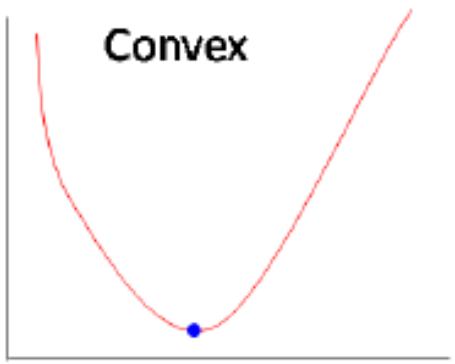
NonConvex

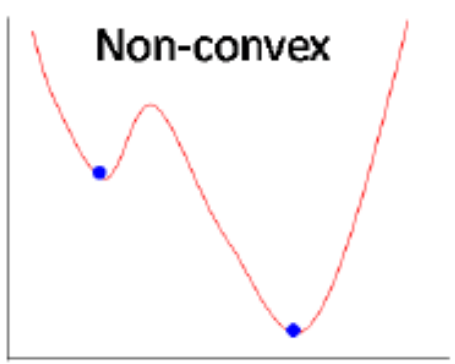


Convex Polygons



Non-convex Polygons





Key Difference: Convex refers to a curvature that extends outwards, whereas non-convex refers to a curvature that extends inward. Non-convex is also referred to as concave.

Convex and non-convex both define the types of curvature. Convex defines the curvature that extends outwards or bulges out. On the other hand, non-convex defines a curvature that extends or bends inward. Thus, the extension of the curve is used to differentiate between the two forms. Convex and non-convex are often used as adjectives to define the entities associated with the shape or curve defined by them.

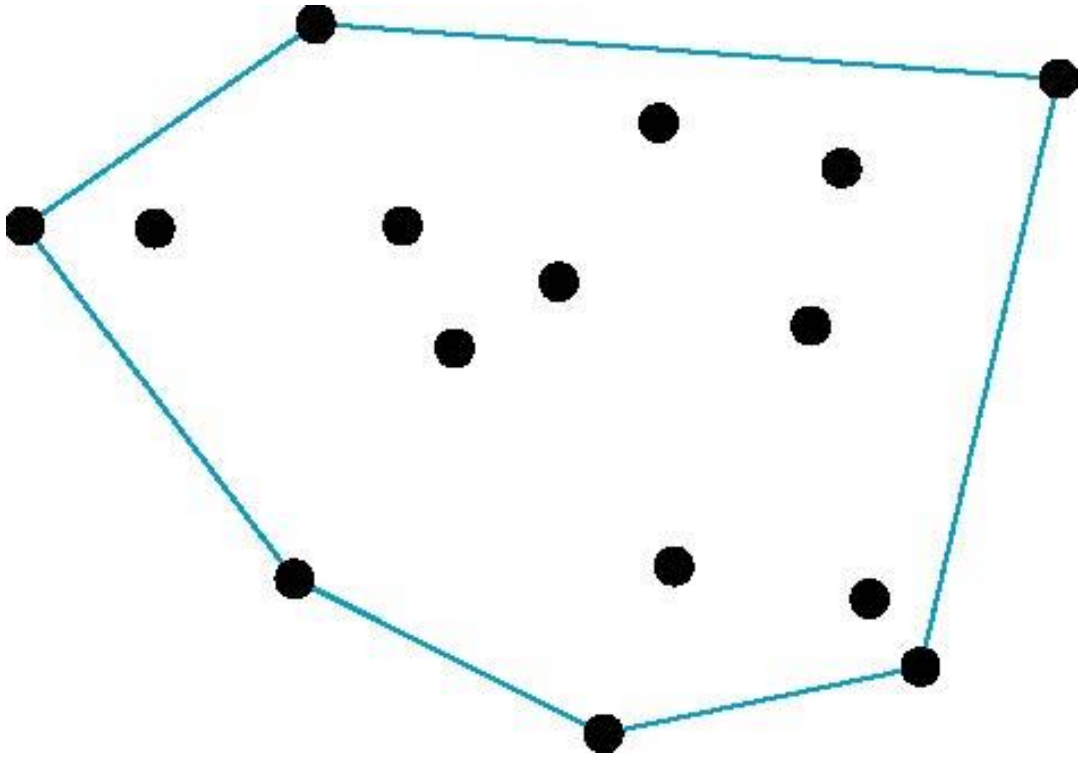
For example, in terms of a polygon, two general categories include convex and non-convex polygons. A convex polygon has no internal angle greater than 180 degrees. However, if a polygon exists with one or more internal angles greater than 180 degrees, then the polygon is known as the non-convex or concave polygon.

In Euclidean space, an object is convex if for every pair of points lying in the object, every point on the straight line segment that joins them also falls within the object. However, if any line segment falls outside the shape or set, then it is regarded to be non-convex. A solid cube is an example of convex, whereas a crescent shape is non-convex (concave).

http://www.differencebetween.info/difference-between-convex-and-non-convex

Convex Hull

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



Affine Sums

- Consider the "sum"

$P=\alpha_1 P_1+\alpha_2 P_2+\ldots..+\alpha_n P_n$

Can show by induction that this sum makes sense iff

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

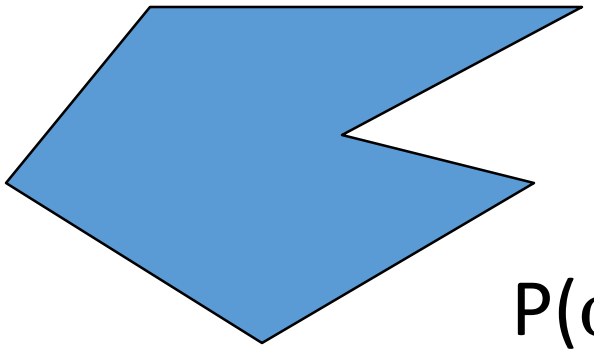
in which case we have the *affine sum* of the points $P_1,P_2,.....P_n$

- If, in addition, $\alpha_i>=0$, we have the *convex hull* of $P_1,P_2,.....P_n$

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

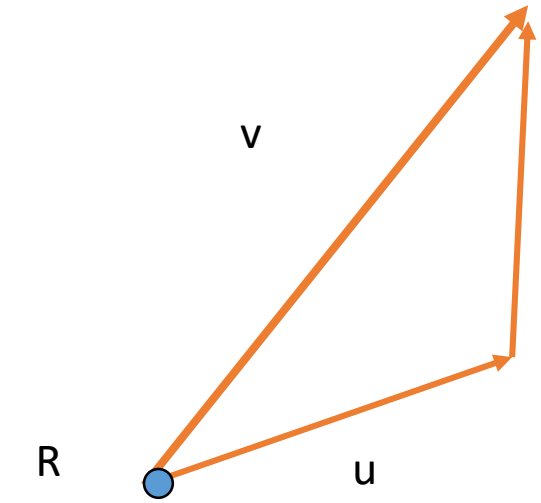
$P(\alpha)$



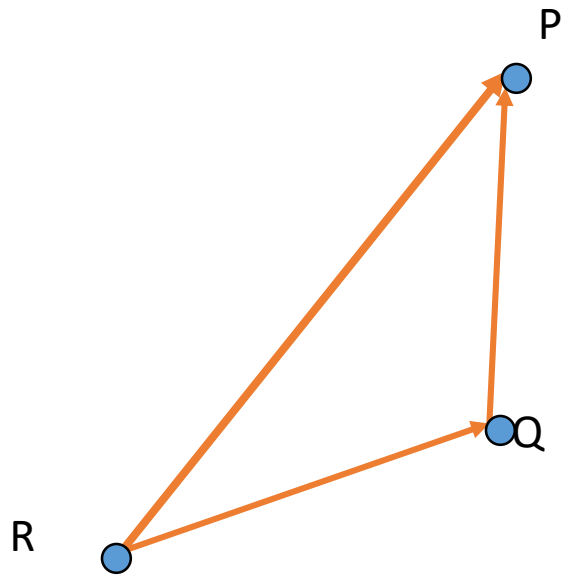
$P(\alpha, \beta)$

Planes

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

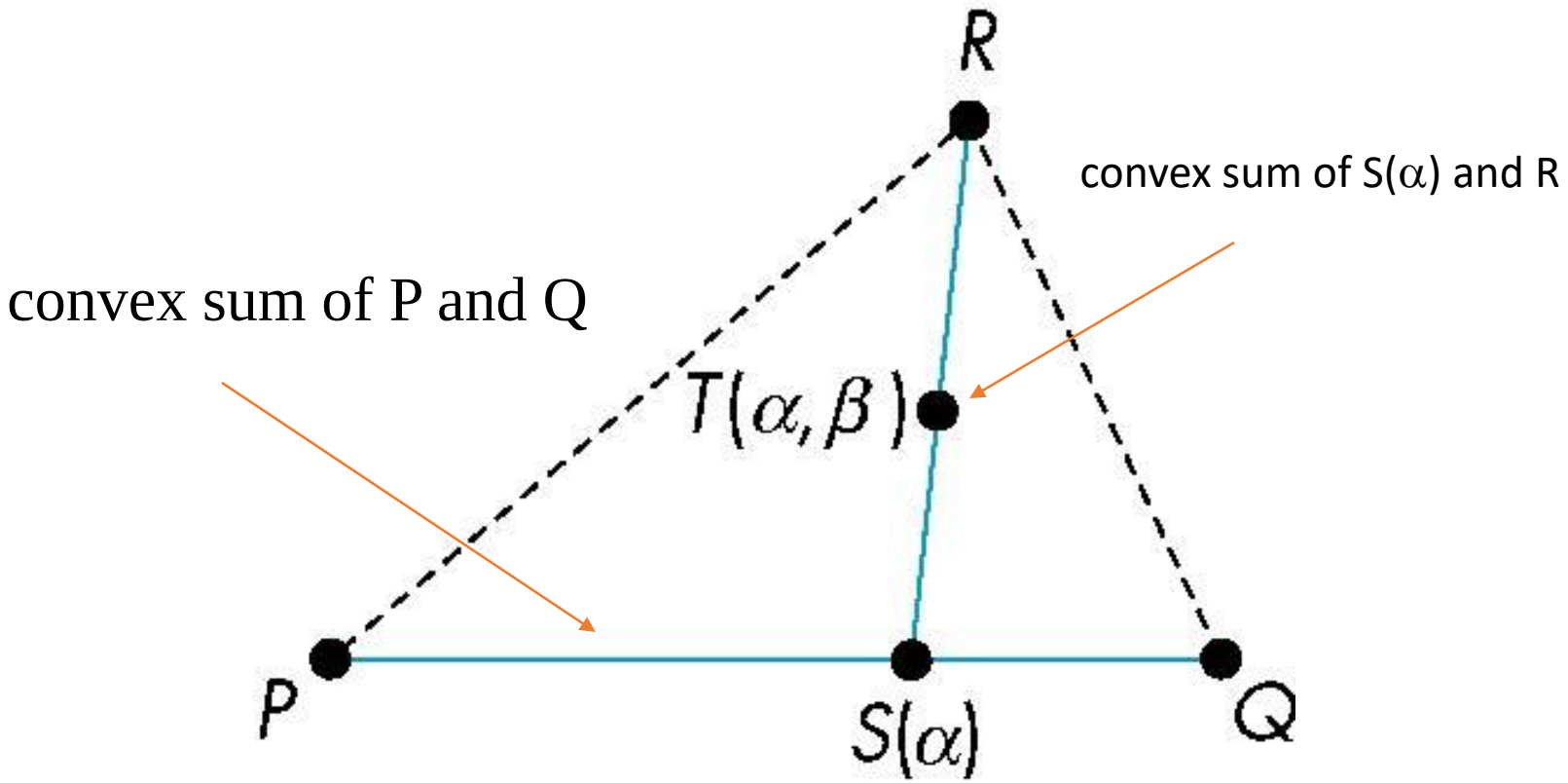


$P(\alpha,\beta)=R+\alpha u+\beta v$



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

Triangles

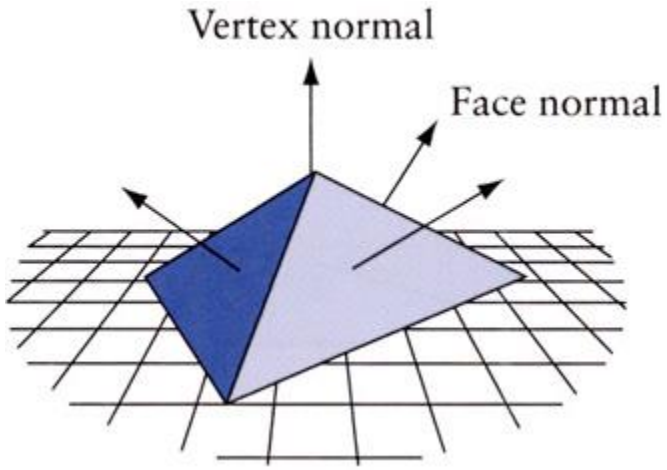


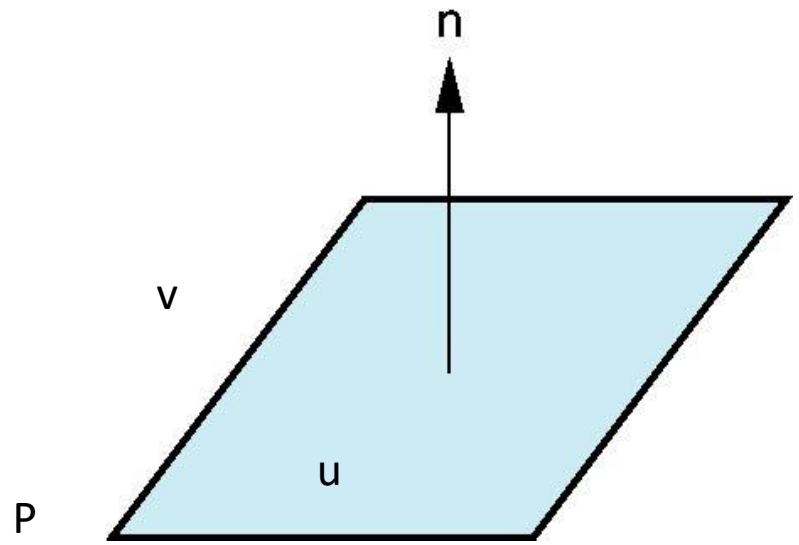
for 0<=α,β<=1, we get all points in triangle

Normals

- Every plane has a vector n normal (perpendicular, orthogonal) to it
- From point-two vector form $P(\alpha,\beta)=R+\alpha u+\beta v$, we know we can use the cross product to find $n = u \times v$ and the equivalent form

$$(P(\alpha)-P) \cdot n=0$$





http://flylib.com/books/en/2.451.1.14/1/

Why we calculate normals

- To make sure light are heading towards the right way
- To make the object feels as much as real world things
- To adapt into different computer settings/etc