UCLA

CS174A : Introduction to Computer Graphics

Kinsey 1240
MW 4-6pm

Scott Friedman, Ph.D
UCLA Institute for Digital Research and Education

Picking and Selection

- Using the mouse to select an item.
- There are several methods available.
- The basic ones
  - The old glSelect() mechanism
  - Color buffer picking
- More advanced
  - Occlusion query extension
  - Ray casting

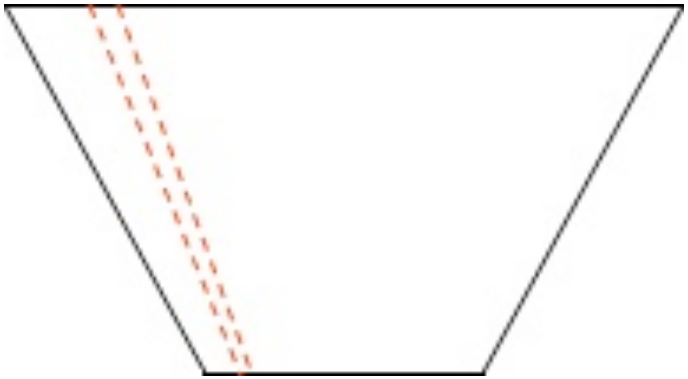
Picking and Selection

- glSelect()
  - Has been in OpenGL from version 1.0
    - Is very slow
      - Implemented in software inside OpenGL library
      - Stalls the hardware rendering pipeline
    - Is depreciated after version 3.0 (basically gone now)
  - The basic idea is simple
    - Use integers to “name” objects in the scene
    - Mark the stream of geometry with those “names”

Picking and Selection

- glSelect()
  - Works by rendering the scene in a special OpenGL mode.
  - A buffer is allocated the collects “hits” (hit buffer)
  - A “hit” is any “name” intersecting the view volume.
  - The function gluPickMatrix() helps by restricting the projection to an area around the mouse click. (limits what needs to be drawn)
    - Say, an area of 4x4 pixels to introduce some fuzz.
    - Tiny frustum from COP through tiny viewport around mouse.
  - Any primitives rendered *after* a “name” object has been defined is placed into a predefined hit buffer and assigned “name”.
  - The “name” objects in the hit buffer tell us what was under the mouse.

Picking and Selection



```
void gluPickMatrix(GLdouble x, GLdouble y, GLdouble dx, GLdouble dy, GLint viewport[4])
{
  if (dx <= 0 || dy <= 0) { return; }
  /* Translate and scale the picked region to the entire window */
  glTranslatef((viewport[2] - 2 * (x - viewport[0])) /
                              dx, (viewport[3] - 2 * (y - viewport[1])) / dy, 0);
  glScalef(viewport[2] / deltax, viewport[3] / dy, 1.0);
}
```

Picking and Selection

- Color Buffer Picking
  - Much simpler and faster.
  - Here an additional, simplified, rendering pass is made.
  - Every object we wish to identify is assigned a unique color.
    - Very simple fragment shader – directly assign color.
  - No lighting or texture mapping performed.
  - Rendered into picking frame buffer.
  - Otherwise scene rendered normally.
    - Offscreen frame buffer. (actually a texture)
    - Z-buffer on.
    - Only draw objects we wish to consider for picking.
      - » Again, no textures, lighting, etc.

Picking and Selection

- Color Buffer Picking
  - Once picking render pass is complete.
  - We *do not* go on to the next frame! (requestAnimationFrame)
    - We are still on the same frame!
  - Call gl.readPixels() to recover result from picking buffer.
    - Using mouse/window x, y position.
  - The pixel color value retrieved corresponds to the object underneath the mouse. (obviously you have to keep track)
  - Use the z-buffer to identify object's position in 3D space
    - Not necessary of you just want to the know the object picked.
    - Slower because z-buffer is cleared before *each* pickable object is drawn
    - Ex: http://xeolabs.com/articles/scenejs-ray-picking-technique

Picking and Selection

- Color Buffer Picking
  - Further enhancements/thoughts
    - Rendering only bounding volumes for scene objects.
    - Only need to render objects considered for picking.
    - Use a restricted frustum (like old style gluPickMartix)
    - Can encode whatever you like into the ”color” bits.
  - Careful when you call readPixels in WebGL.
    - Your GPU *may* still be busy drawing you could get zeros back.
      - » WebGL/OpenGL are asynchronous with the CPU.
      - » All depends on the implementation by browser.
    - You can create your context with `preserveDrawingBuffer: true`
      - » Or call `gl.flush()` if this happens. (see API docs)
      - » Which may have performance consequences

```
// Reading back color value

gl.readPixels( mouseX, mouseY, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, buffer);
```

Picking and Selection

- Color Buffer Picking
  - Implementation
    - Use the normal color buffer before regular rendering?
      - Must keep all your rendering for a single frame within the browser event
        - » e.g. RequestAnimationFrame()
        - » Once the callback returns the browser assumes you are 'done' and will composite the result with the rest of the browser window.
        - » This is essentially the swapBuffers() call in regular OpenGL
    - Probably would work
      - Here you keep your rendering within the onMouseDown event
        - » May have some issues with color values rounding…

```
// Reading back color value

gl.readPixels( mouseX, mouseY, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, buffer);
```

Picking and Selection

- Color Buffer Picking
  - Implementation
    - Use the alpha channel?
      - Limited number of objects can be represented as pickable.
      - One pass
      - No blending
        - » Or maybe yes blending
        - » Fragment shader would have to know how to interpret alpha bits to only use bits representing transparency and modify the source alpha to remove the picking index before the hardware blends it.

```
// Reading back color value

gl.readPixels( mouseX, mouseY, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, buffer);
```

Picking and Selection

- Color Buffer Picking
  - Implementation
    - Typically you will see an offscreen rendering buffer created.
    - Picking color index values written to this buffer and read back.
    - Avoids issues that come with multi-sampling/antialiasing in the primary color buffer.
    - Quite a bit more work to setup
      - Requires keeping correct buffers bound when rendering

```
// Reading back color value

gl.readPixels( mouseX, mouseY, 1, 1, gl.RGBA, gl.UNSIGNED_BYTE, buffer);
```

Picking and Selection

```
pickBuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, pickBuffer );
pickBuffer.width  = 512; // These should match your canvas
pickBuffer.height = 512;
pickTexture = gl.createTexture();
gl.bindTexture( gl.TEXTURE_2D, pickTexture );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR );
gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGBA, pickBuffer.width,
          pickBuffer.height, 0, gl.RGBA, gl.UNSIGNED_BYTE, null );
depthBuffer = gl.createRenderbuffer();
gl.bindRenderbuffer( gl.RENDERBUFFER, depthBuffer );
gl.renderbufferStorage( gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
          pickBuffer.width, pickBuffer.height );
gl.framebufferTexture2D( gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
          gl.TEXTURE_2D, pickTexture, 0 );
gl.framebufferRenderbuffer( gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
          gl.RENDERBUFFER, depthBuffer );
// Reset for normal rendering
gl.bindTexture( gl.TEXTURE_2D, null );
gl.bindRenderbuffer( gl.RENDERBUFFER, null );
gl.bindFramebuffer( gl.FRAMEBUFFER, null );
```

Picking and Selection

```
pickBuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, pickBuffer );
pickBuffer.width  = 512; // These should match your canvas
pickBuffer.height = 512;
pickTexture = gl.createTexture();
gl.bindTexture( gl.TEXTURE_2D, pickTexture );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR );
gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGBA, pickBuffer.width,
          pickBuffer.height, 0, gl.RGBA, gl.UNSIGNED_BYTE, null );
depthBuffer = gl.createRenderbuffer();
gl.bindRenderbuffer( gl.RENDERBUFFER, depthBuffer );
gl.renderbufferStorage( gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
          pickBuffer.width, pickBuffer.height );
gl.framebufferTexture2D( gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
          gl.TEXTURE_2D, pickTexture, 0 );
gl.framebufferRenderbuffer( gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
          gl.RENDERBUFFER, depthBuffer );
// Reset for normal rendering
gl.bindTexture( gl.TEXTURE_2D, null );
gl.bindRenderbuffer( gl.RENDERBUFFER, null );
gl.bindFramebuffer( gl.FRAMEBUFFER, null );
```

Picking and Selection

```
pickBuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, pickBuffer );
pickBuffer.width  = 512; // These should match your canvas
pickBuffer.height = 512;
pickTexture = gl.createTexture();
gl.bindTexture( gl.TEXTURE_2D, pickTexture );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR );
gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGBA, pickBuffer.width,
          pickBuffer.height, 0, gl.RGBA, gl.UNSIGNED_BYTE, null );
depthBuffer = gl.createRenderbuffer();
gl.bindRenderbuffer( gl.RENDERBUFFER, depthBuffer );
gl.renderbufferStorage( gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
          pickBuffer.width, pickBuffer.height );
gl.framebufferTexture2D( gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
          gl.TEXTURE_2D, pickTexture, 0 );
gl.framebufferRenderbuffer( gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
          gl.RENDERBUFFER, depthBuffer );
// Reset for normal rendering
gl.bindTexture( gl.TEXTURE_2D, null );
gl.bindRenderbuffer( gl.RENDERBUFFER, null );
gl.bindFramebuffer( gl.FRAMEBUFFER, null );
```

Picking and Selection

```
pickBuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, pickBuffer );
pickBuffer.width  = 512; // These should match your canvas
pickBuffer.height = 512;
pickTexture = gl.createTexture();
gl.bindTexture( gl.TEXTURE_2D, pickTexture );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR );
gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGBA, pickBuffer.width,
          pickBuffer.height, 0, gl.RGBA, gl.UNSIGNED_BYTE, null );
depthBuffer = gl.createRenderbuffer();
gl.bindRenderbuffer( gl.RENDERBUFFER, depthBuffer );
gl.renderbufferStorage( gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
          pickBuffer.width, pickBuffer.height );
gl.framebufferTexture2D( gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
          gl.TEXTURE_2D, pickTexture, 0 );
gl.framebufferRenderbuffer( gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
          gl.RENDERBUFFER, depthBuffer );
// Reset for normal rendering
gl.bindTexture( gl.TEXTURE_2D, null );
gl.bindRenderbuffer( gl.RENDERBUFFER, null );
gl.bindFramebuffer( gl.FRAMEBUFFER, null );
```

Picking and Selection

```
pickBuffer = gl.createFramebuffer();
gl.bindFramebuffer( gl.FRAMEBUFFER, pickBuffer );
pickBuffer.width  = 512; // These should match your canvas
pickBuffer.height = 512;
pickTexture = gl.createTexture();
gl.bindTexture( gl.TEXTURE_2D, pickTexture );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR );
gl.texParameteri( gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR );
gl.texImage2D( gl.TEXTURE_2D, 0, gl.RGBA, pickBuffer.width,
          pickBuffer.height, 0, gl.RGBA, gl.UNSIGNED_BYTE, null );
depthBuffer = gl.createRenderbuffer();
gl.bindRenderbuffer( gl.RENDERBUFFER, depthBuffer );
gl.renderbufferStorage( gl.RENDERBUFFER, gl.DEPTH_COMPONENT16,
          pickBuffer.width, pickBuffer.height );
gl.framebufferTexture2D( gl.FRAMEBUFFER, gl.COLOR_ATTACHMENT0,
          gl.TEXTURE_2D, pickTexture, 0 );
gl.framebufferRenderbuffer( gl.FRAMEBUFFER, gl.DEPTH_ATTACHMENT,
          gl.RENDERBUFFER, depthBuffer );
// Reset for normal rendering
gl.bindTexture( gl.TEXTURE_2D, null );
gl.bindRenderbuffer( gl.RENDERBUFFER, null );
gl.bindFramebuffer( gl.FRAMEBUFFER, null );
```

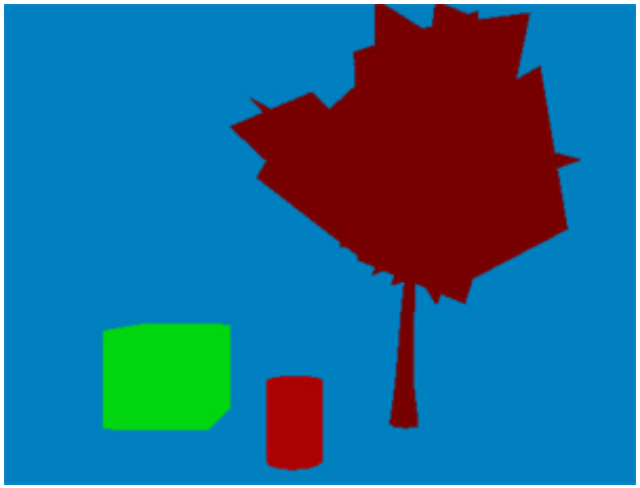
Picking and Selection

```
// To change render target
gl.bindFramebuffer( gl.FRAMEBUFFER, pickBuffer );
gl.viewport( ... );
gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT );
drawForPicking();

// Reset for normal rendering
gl.bindFramebuffer( gl.FRAMEBUFFER, null );
gl.viewport( ... );
gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT );
drawNormally();
```

Picking and Selection

- Color Buffer Picking
  - What the picking color buffer looks like
  - Notice how transparency can pose a problem?



Picking and Selection

- Other techniques
  - Occlusion query
    - We want to know if something will be visible **before** we render it.
      - » Why would we want to do this?
    - Reading z-values instead of color.
    - Allows determining order of objects under mouse
    - Harder to do region queries (e.g. rubber band box)
  - Used for advanced scene culling
    - Depth peeling
  - Hardware accelerated 2d collision detection

  - Unfortunately, WebGL 1.0 does not support the feature
    - Needs to use extensions
    - 2.0 will support basic functionality

Collision Detection

- A large a complex topic
  - Object Representations and Bounding Volumes.
  - Simple Intersection Tests
  - Bounding Volume Hierarchies.
  - Interested in…
    - Whether object A has collided with object B.

Collision Detection

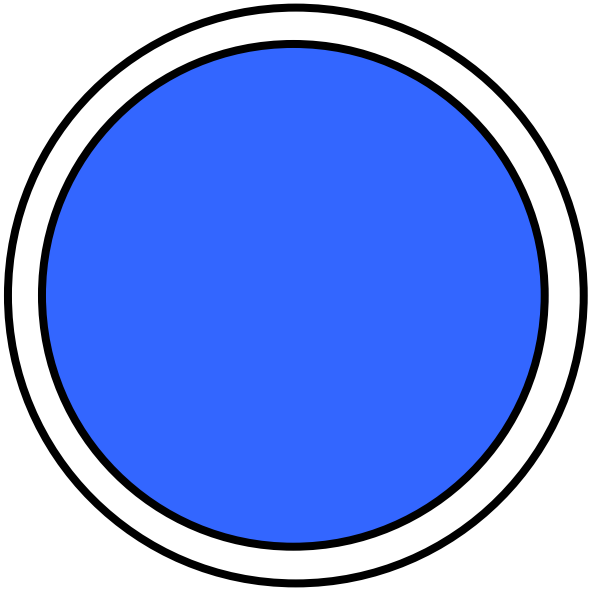
- Object Representation
  - We perform collision detection by computing whether one primitive intersects another.
  - We could do this by comparing all triangles in a scene with each other.
    - This would be slow and produce a lot of useless results.
  - Stick to objects in scene we care about.
    - Still checking objects that do not collide with anything.
    - Say we have ten objects that potentially hit each other made up of 100 triangles each
    - Several tens of thousands of comparisons required!
  - Need to be smarter about this…

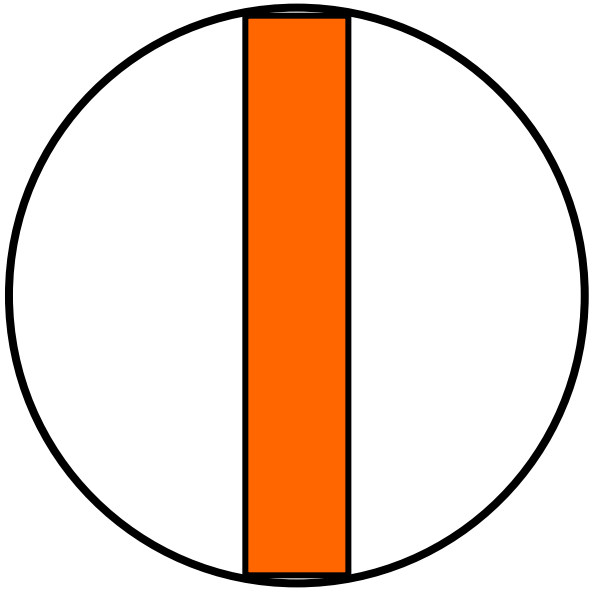
Collision Detection

- Object Representation
  - Has to be a better way
  - There is, it is called a bounding volume.
    - A sphere is the simplest.
      - Not an exact representation, but…
      - Now we only need less than 50 comparisons! (45)
      - If two spheres intersect a more thorough check can then be performed.
        - » We spend time only when we need to
      - Or, test itself can be sufficient if objects are far enough away and close inspection is not necessary.
        - » Whatever is appropriate for the problem.

Collision Detection

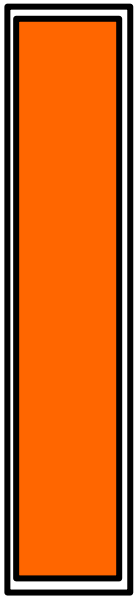
- Object Representation
  - A good fit is desirable.
  - Spheres, unfortunately, are not always a good choice.





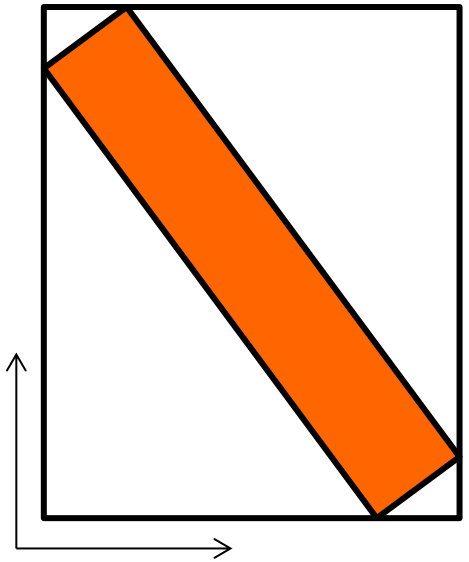
Collision Detection

- Object Representation
  - Sometimes a bounding rectangle would be better.
  - More complex to perform intersection tests, however.
  - Probably not that great for a sphere.



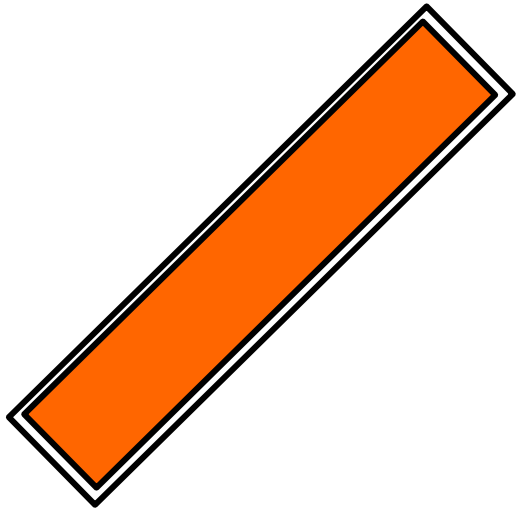
Collision Detection

- Object Representation
  - Sometimes even a bounding rectangle doesn't work.
  - Simplest box is called an axis-aligned bounding box AABB
    - Box edges aligned with primary axes.



Collision Detection

- Object Representation
  - An oriented bounding box may be best.
  - *Even more* complex to perform intersection tests, however.
  - As you can see – there are tradeoffs to be considered.



Collision Detection

- Object Representation
  - Which simplified bounding shape to use depends on how accurate you need to be and speed.
  - Whichever one you use there is always a need to determine distance. Which in 3D is computed using…

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

  - Hacker 101: When only *comparing* distances we can avoid the square root operation for speed!
    - e.g. is this closer than that

Collision Detection

- Object Representation
  - Desirable Properties
    - *Inexpensive* Intersection Tests
    - Tight Fitting
    - Inexpensive to Compute Bounding Volume
    - Easy to Rotate and Transform (oh yes…)
    - Uses minimal memory
  - Generally, want to use inexpensive tests before resorting to more expensive tests.

Collision Detection

- Axis Aligned Bounding Box
  - Simple to compute
  - Several ways to represent them
    - Min / Max extreme points (six floats)
    - Min point and extents (six floats or three floats + 3 half)
    - Center point and half-widths (same as above)
      - » Last two are most efficient memory wise.
  - Computationally,
    - Min-extents is slowest, requiring the most operations.
  - Building an AABB is straightforward regardless.

Collision Detection

- Axis Aligned Bounding Box
  - Example, intersection of two AABB

```
struct AABB {
  Point min;
  Point max;
};

int testAABB( AABB a, AABB b )
{
  // Exit if separated along an axis
  if ( a.max[0] < b.min[0] || a.min[0] > b.max[0] ) return 0;
  if ( a.max[1] < b.min[1] || a.min[1] > b.max[1] ) return 0;
  if ( a.max[2] < b.min[2] || a.min[2] > b.max[2] ) return 0;
  // Overlapping on all axes means there is an intersection
  return 1;
}
```

Collision Detection

- Bounding Sphere
  - Simple intersection test…

```
struct Sphere {
  Point c;
  float r;
};

int testSphere( Sphere a, Sphere b )
{
  // Calculate squared distance between centers
  Vector d = a.c – b.c;
  float dist2 = Dot( d, d );
  // Spheres intersect if squared distance is less than
  //   squared sum of radii
  float radiusSum = a.r + b.r;
  return dist2 <= radiusSum * radiusSum;
}
```

Collision Detection

- Bounding Sphere – be careful
  - Computing the bounding sphere itself, **not so simple**…
  - A naïve brute force algorithm runs in **$O(n^5)$!**
  - Ritter, describes an iterative algorithm.
    - Start with a sphere based on two points.
    - Iteratively add points to grow.
    - Does reasonably well except
      - » Quality is sensitive to the order points are added.
  - Welzl, uses computational geometry to achieve an expected $O(n^1)$ time. ☺
    - Implementation is complex, however. ☹

Collision Detection

- Bounding Sphere - Ritter

```
void MostSeparatedPointsPointsOnAABB( int &min, int &max, Point pt[], int
numPts )
{
  // find most extreme points
  int minx = 0, maxx = 0, miny = 0, maxy = 0, minz = 0, maxz = 0;
  for ( int i=1; i<numPts; i++ )
  {
    if ( pt[i].x < pt[minx].x ) minx = i;
    if ( pt[i].x > pt[maxx].x ) maxx = i;
    if ( pt[i].y < pt[miny].x ) miny = i;
    if ( pt[i].y > pt[maxy].x ) maxy = i;
    if ( pt[i].z < pt[minz].x ) minz = i;
    if ( pt[i].z > pt[maxz].x ) maxz = i;
  }
  // compoute squared distances for three sets of points
  float dist2x = Dot( pt[maxx] – pt[minx], pt[maxx] – pt[minx] );
  float dist2y = Dot( pt[maxy] – pt[miny], pt[maxy] – pt[miny] );
  float dist2z = Dot( pt[maxz] – pt[minz], pt[maxz] – pt[minz] );

Continued...
```

Collision Detection

- Bounding Sphere - Ritter

```
  // pick the most distant pair
  min = minx;
  max = maxx;
  if ( dist2y > dist2x && dist2y > dist2z )
  {
    max = maxy;
    min = miny;
  }
  if ( dist2z > dist2x && dist2z > dist2z )
  {
    max = maxz;
    min = minz;
  }
}

void SphereFromDistantPoints( Sphere &s, Point p[], int numPts )
{
  // find the most separated point pair encompassing the AABB
  int min, max;
  MostSeparatedPointsPointsOnAABB( min, max, pt, numPts );
  // setup sphere to emcompass just these two points
  s.c = ( pt[min] + pt[max] ) * 0.5;
  s.r = Dot( pt[max] – s.c, pt[max] – s.c );
  s.r = Sqrt( s.r );
}
```

Collision Detection

- Bounding Sphere - Ritter

```
// given sphere s and point p, update s to just encompass p
void SphereOfSphereAndPt( Sphere &s, Point &p )
{
  // compute squared distance between point and sphere center
  Vector d = p – s.c;
  float dist2 = Dot( d, d );
  // only update s if point p is outside it
  if ( dist2 > s.r * s.r ) {
    float dist – sqrt( dist2 );
    float newRadius = ( s.r + dist ) * 0.5f;
    float k = (newRadius – s.r) / dist;
    s.r = newRadius;
    s.c += d * k;
  }
}

Void RitterSphere( Sphere &s, Point pt[], int numPts )
{
  // get sphere encompassing two approximately most distant pts
  SphereFromDistantPoints( s, pt, numPts );
  // grow sphere to include all points
  for ( int i=0; i<numPts; i++ )
    SphereOfSphereAndPr( s, pt[i] );
}
```

Collision Detection

- Processing
  - Need a data structure to hold BVs
  - Organization depends on application
    - Maybe all you care about is your spaceship hitting an asteroid?
    - Then all you want to check is the spaceship against the asteroids.
    - You would not bother with asteroid/asteroid checks.
      - » Just like the real "Asteroids"
  - Advanced – what about hierarchies of bounding volumes?
    - Could be a good thing – eliminating large groups of objects
    - If represented objects move these bounding volumes of bounding values have to be updated to account for movement.

Collision Detection

- Processing
  - There are many types of bounding objects
  - Spheres and AABB are suitable for the project
    - Others get more complex - not worth it with the time you have available.
  - You may also want to sphere against a plane
    - Useful to know if eye/camera has hit something
    - Again, very simple

Collision Detection

- Processing

```
struct Sphere {
  Point c;
  float r;
};

struct Plane {
  Vector n;     // plane normal, points on plane satisfy
  float d;      // d=Dot(n,p) for a given point p on the plane.
};

int testSpherePlane( Sphere s, Plane p )
{
  // for a normalized plane (|p.n|=1), evaluating the plane equation
  //  for a point gives the signed distance of the point to the plane
  float dist = Dot( s.c, p.n ) – p.d;
  // if sphere center within +/- radius from plane, plane intersects sphere
  return Abs( dist ) < s.r;
}
```

Collision Detection

- Processing
  - Point normal form – compact representation
    - n is normal to plane, P is point on plane, X any other point on place, n dot (X-P) = 0 and d = n dot P

```
struct Plane {  // point normal form
  Vector n;     // plane normal, points on place satisfy
  float d;      // d=Dot(n,p) for a given point p on the plane.
};

Plane computePlane( Point i, Point j, Point k )
{
  // given three non-collinear points (CCW), compute plane equation
  Plane p;
  p.n = Normalize( Cross( j – i, k – i ) );
  p.d = Dot( p.n, i );
  return p;
}
```

Collision Detection

- Picking and Collision detection are each considered advanced topics for your projects.