Computer Graphics: Interaction

Dept. of Game Software
Yejin Kim

Overview

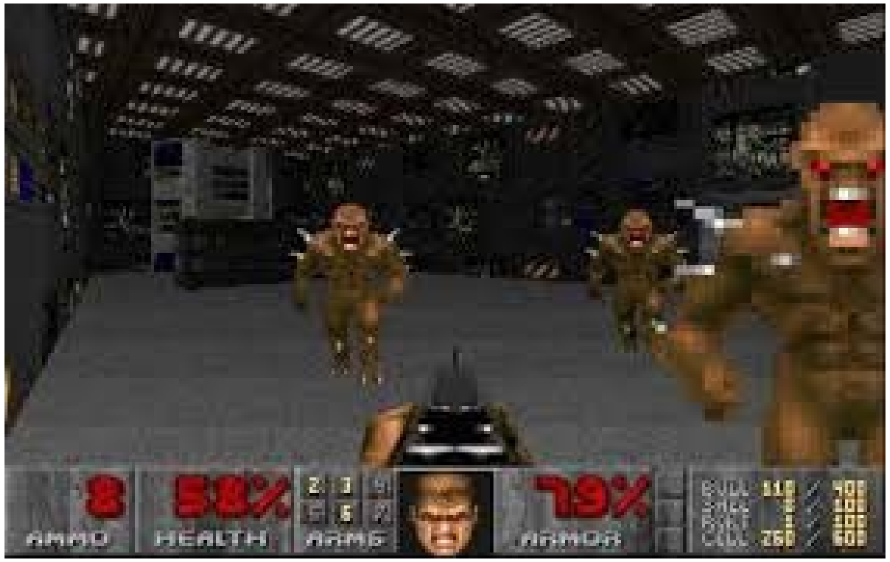
- Picking
- Bounding Volumes
- Collision Detection
- Visibility Culling
- Tutorials

Interactive Graphics

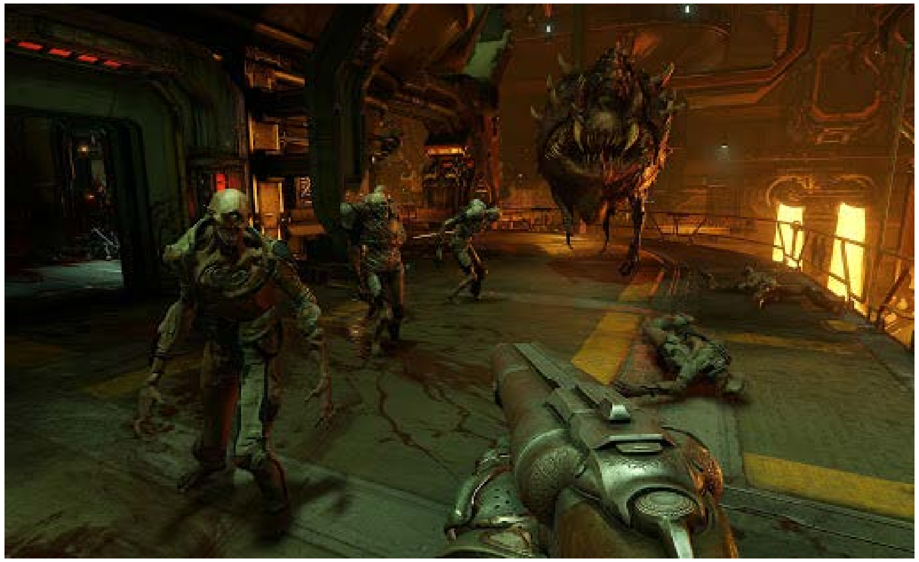
- Interactivity vs Realism

$$\text{Time-faithful (interactivity)} \longleftrightarrow \text{Space-faithful (visual realism)}$$

  - Realism and Interactivity are mutually conflicting goals
  - The complexity of graphics datasets of interest keeps growing

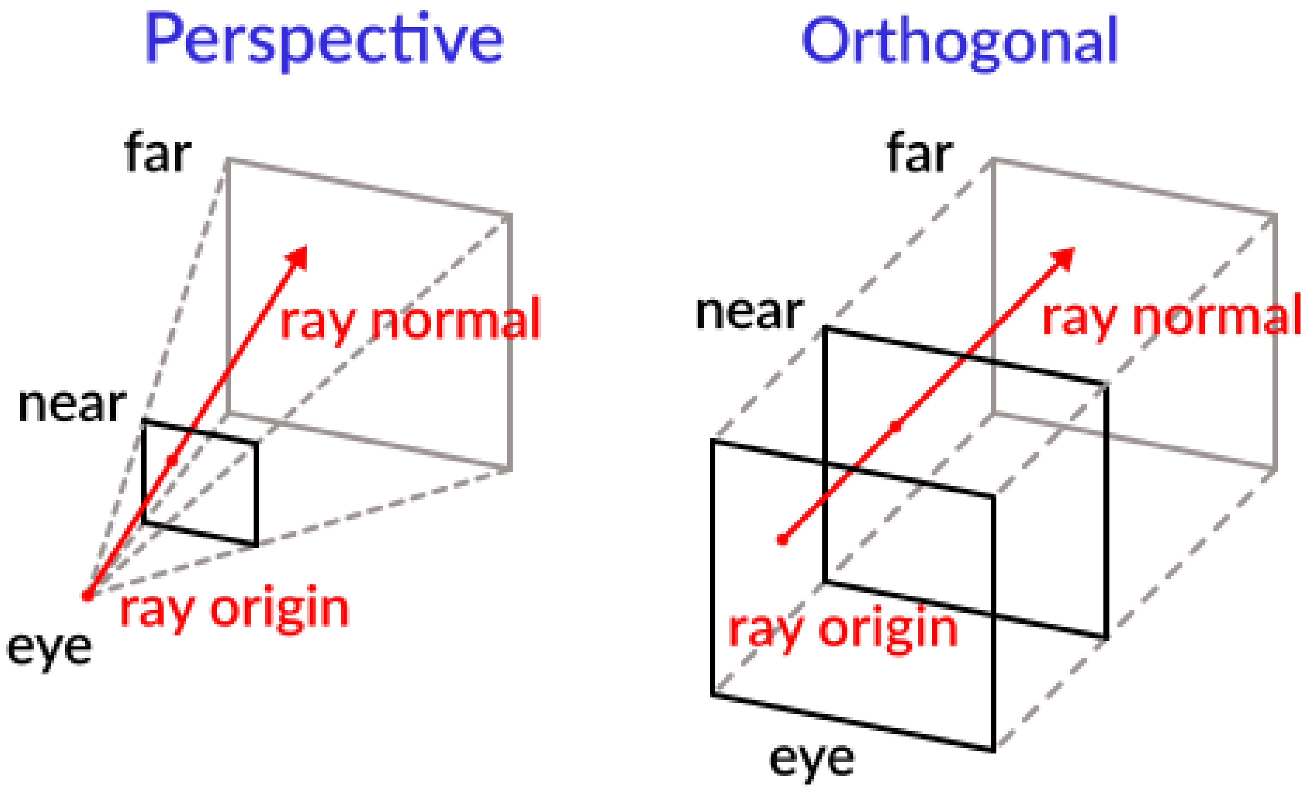


Doom (1993)


Doom 4 (2016)

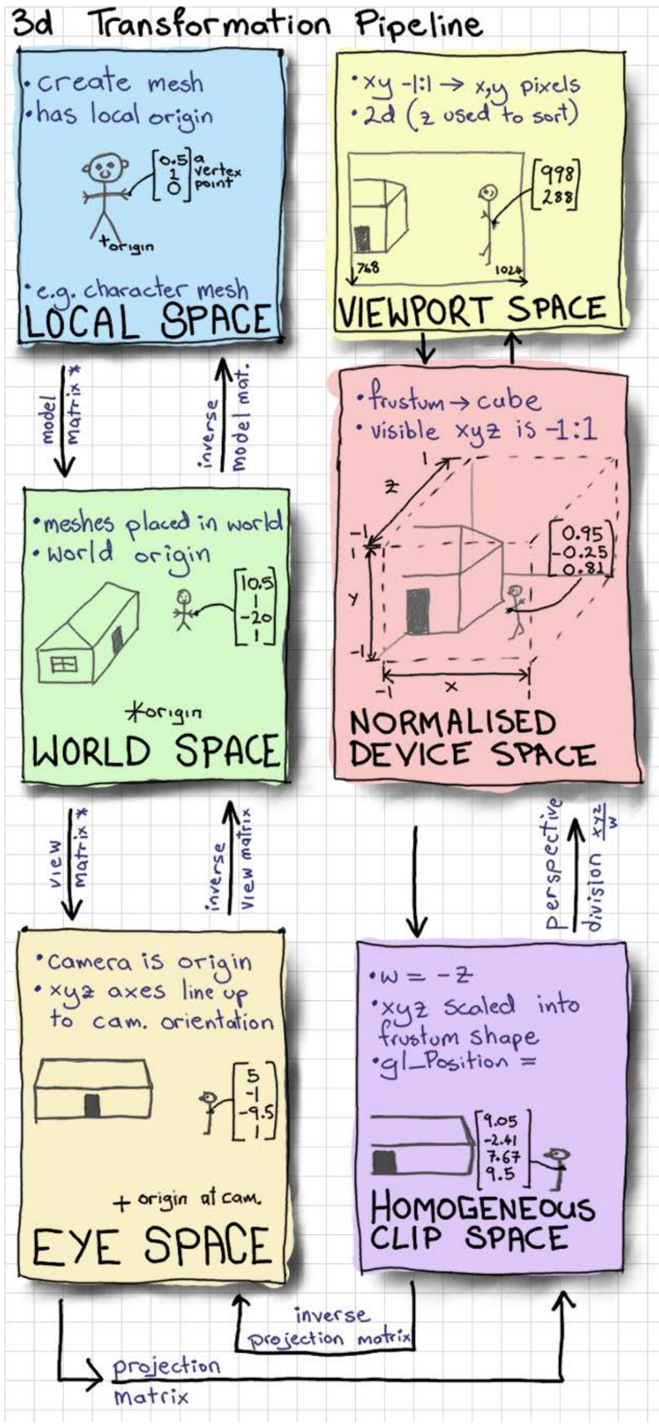
Picking

- Mouse picking with ray casting
  - Mouse picking: Click on, or "pick" a 3d object in a scene using the mouse cursor
  - Ray casting: project(cast) a straight line from a camera's (eye's) position in the direction of the 3D space



Picking

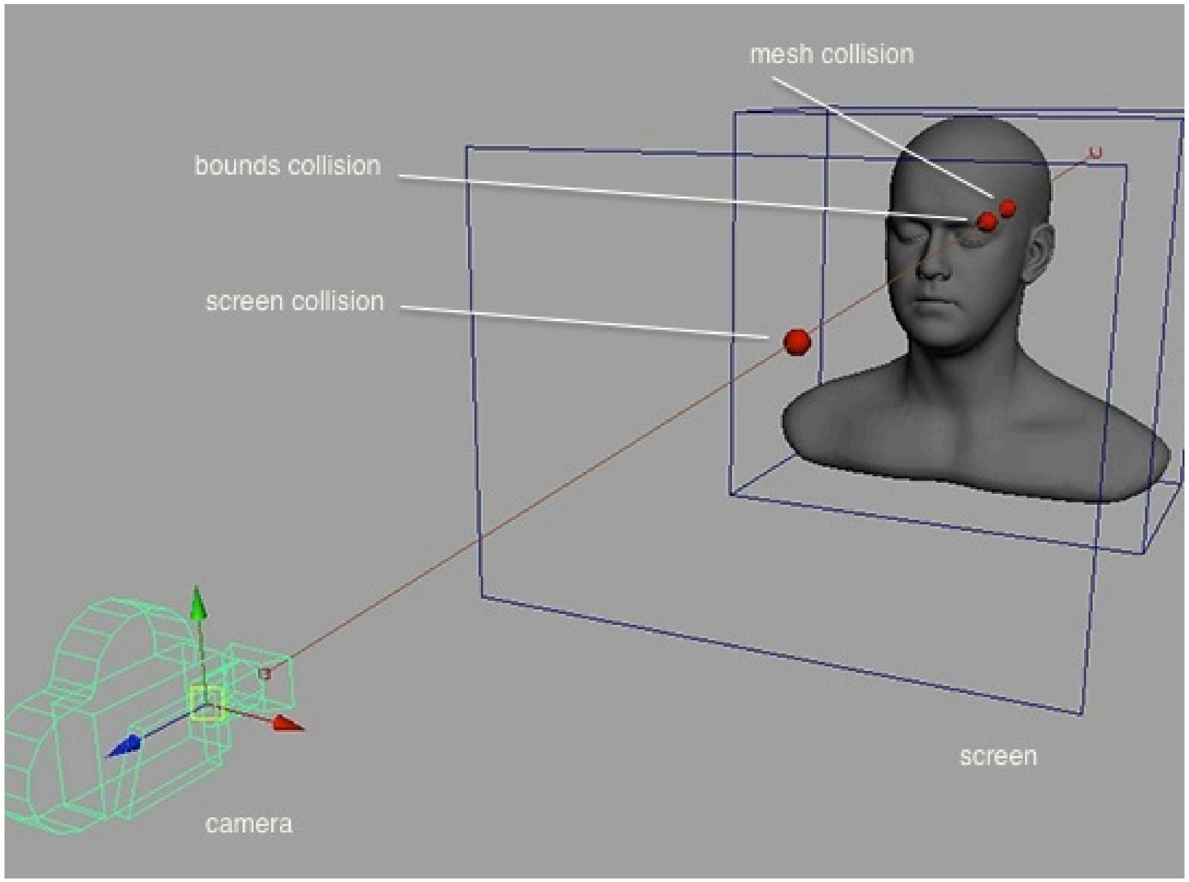
- Picking an object in 3D space:
  1. Get a 2D vector (x, y) of the position of the mouse cursor in **screen space** (the client window in pixels)
  2. Transform the 2D vector representing the mouses position in screen space to a 3D vector (x, y, z) representing a ray in **view space**
  3. Transform the view space ray into a **world space** ray
  4. Transform the model, or objects vertex positions from model (or local) space to world space
  5. Loop through each of the triangles in the model to find out which (if any) triangle the ray intersects with



[Anton Gerdelan, 2016]

Picking

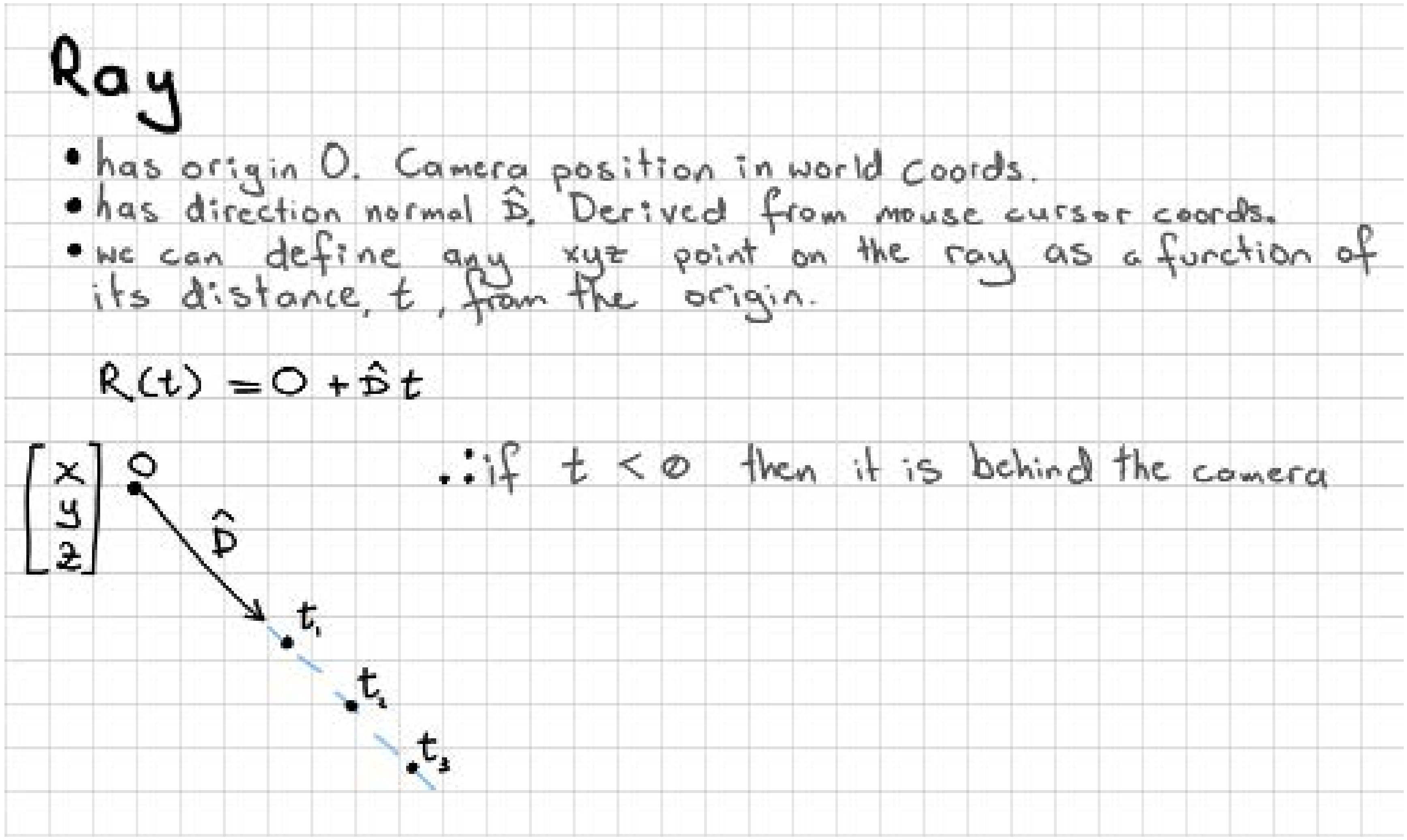
- Ray-triangle intersection
    1. Intersect a ray with a plane
    2. Check if a point is inside triangle



[Away3D, 2012]

Picking

- Ray



Ray

- has origin O. Camera position in world coords.
- has direction normal $\hat{D}$. Derived from mouse cursor coords.
- we can define any xyz point on the ray as a function of its distance, t, from the origin.

$$R(t) = O + \hat{D}t$$

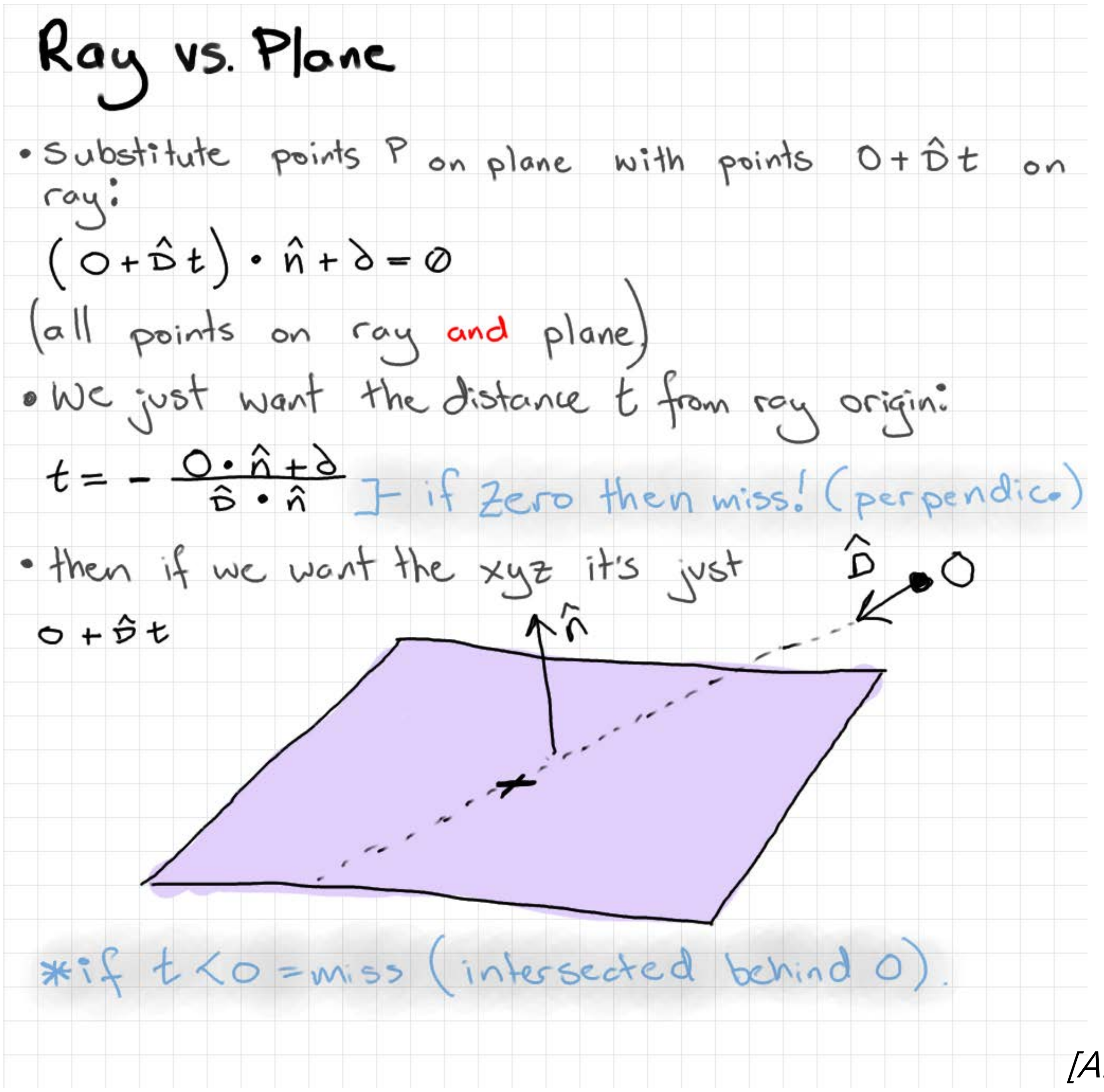
$\therefore$ if $t < 0$ then it is behind the camera

$\begin{bmatrix} x \\ y \\ z \end{bmatrix}$
O
$\hat{D}$
t_1
t_2
t_3

[Anton Gerdelan, 2016]

Picking

- Ray-plane intersection test



Ray vs. Plane

- Substitute points P on plane with points $O + \hat{D}t$ on ray:

$$(O + \hat{D}t) \cdot \hat{n} + d = 0$$

(all points on ray and plane)

- We just want the distance t from ray origin:

$$t = -\frac{O \cdot \hat{n} + d}{\hat{D} \cdot \hat{n}}$$ $\leftarrow$ if zero then miss! (perpendic.)

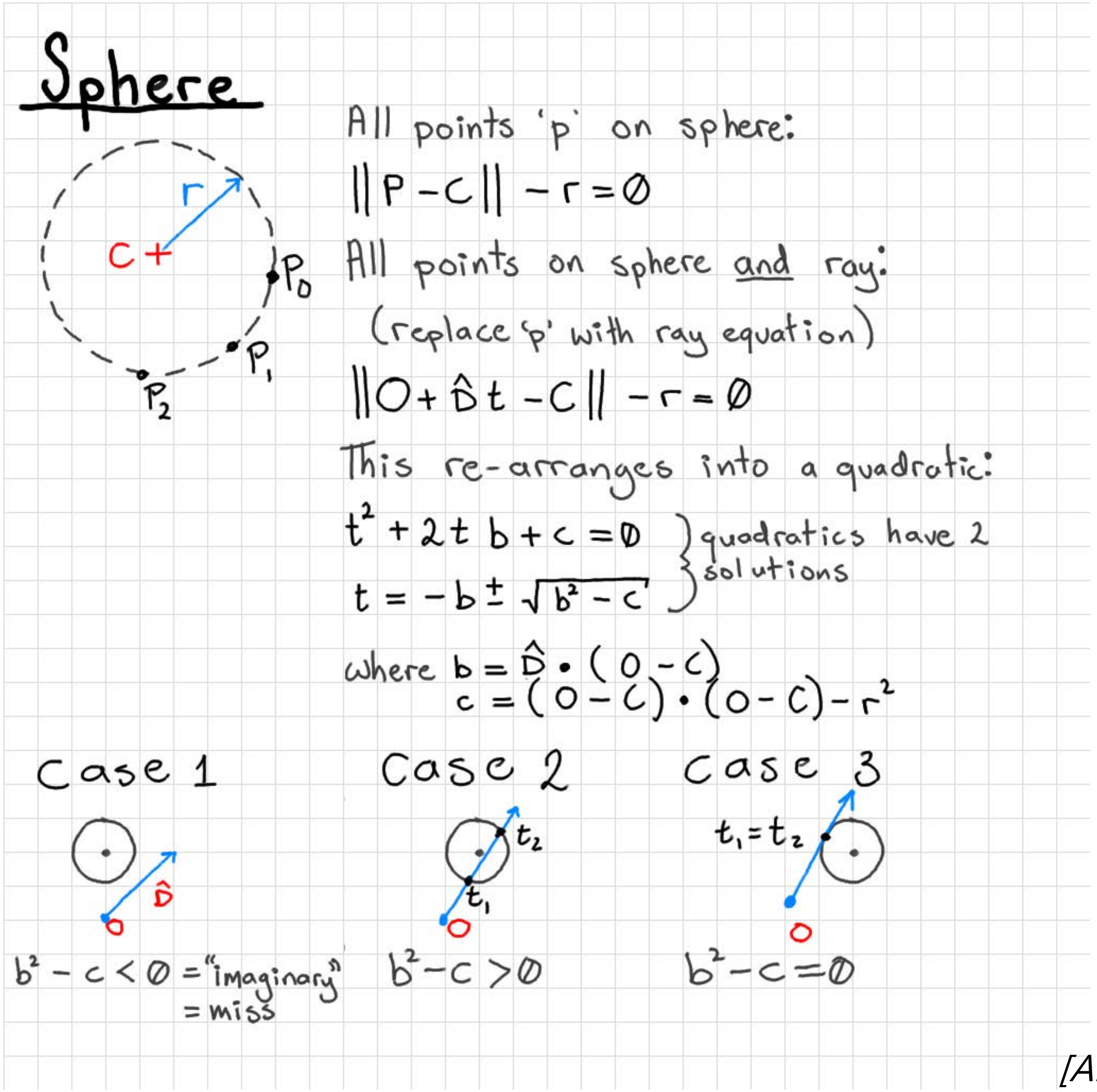
- then if we want the xyz it's just $O + \hat{D}t$

*if $t < 0$ = miss (intersected behind O).

[Anton Gerdelan, 2016]

Picking

- Ray-sphere intersection test



C
r
P_0
P_1
P_2

Sphere

All points 'p' on sphere:

$$\|P - C\| - r = 0$$

All points on sphere <u>and</u> ray:

(replace 'p' with ray equation)

$$\|O + \hat{D}t - C\| - r = 0$$

This re-arranges into a quadratic:

$$t^2 + 2tb + c = 0$$

$$t = -b \pm \sqrt{b^2 - c}$$

quadratics have 2 solutions

where $b = \hat{D} \cdot (O - C)$

$c = (O - C) \cdot (O - C) - r^2$

Case 1

$\hat{D}$
O

$b^2 - c < 0$ = "imaginary" = miss

Case 2

t_2
t_1
O

$b^2 - c > 0$

Case 3

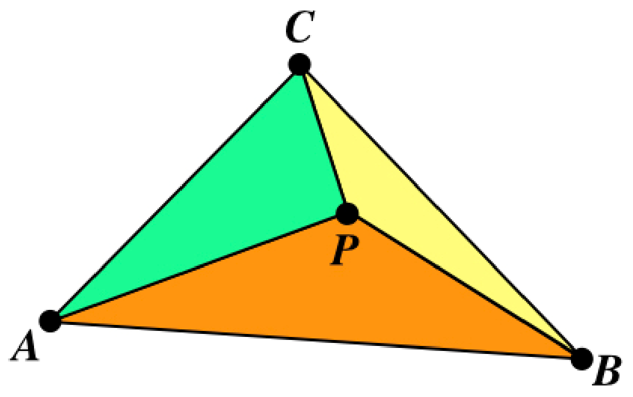
$t_1 = t_2$
O

$b^2 - c = 0$

[Anton Gerdelan, 2016]

Picking

- Barycentric coordinates
  - A coordinate system in which the location of a point is specified by reference to a triangle for pints in a plane
    - Use the cross product of two vectors for the area of a triangle

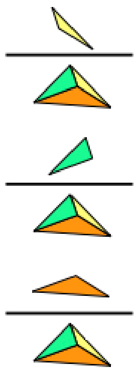


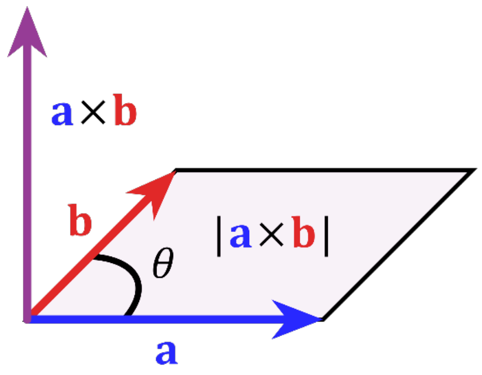
$$\boldsymbol{P} = w_A \times \boldsymbol{A} + w_B \times \boldsymbol{B} + w_C \times \boldsymbol{C}$$

$$w_A = \frac{\Delta PBC}{\Delta ABC} =$$

$$w_B = \frac{\Delta PCA}{\Delta ABC} =$$

$$w_C = \frac{\Delta PAB}{\Delta ABC} =$$





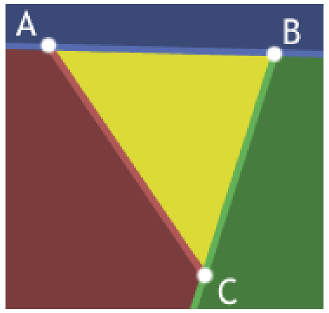
inside condition

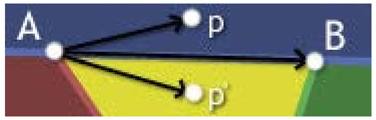
$$0 \le w_A, w_B, w_C \le 1 \qquad w_A + w_B + w_C = 1$$

Picking

- Same side test
  - Test a point is on the correct side of a line

  → Use a direction(sign) of the cross product of two vectors





e.g. the cross product of [B-A] and [p-A] → pointing out

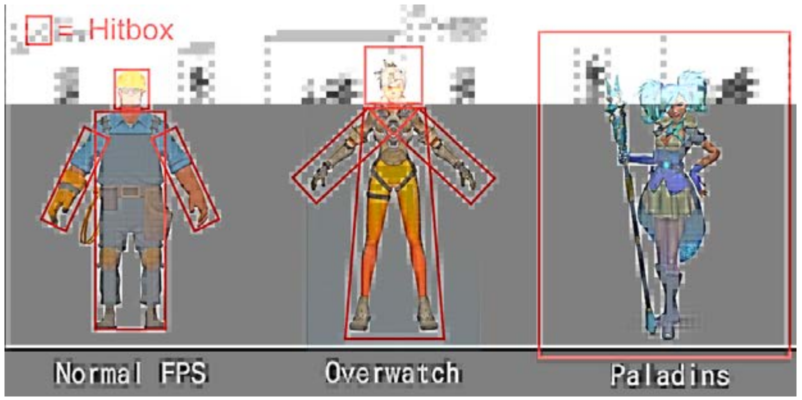
  the cross product of [B-A] and [p'-A] → pointing in

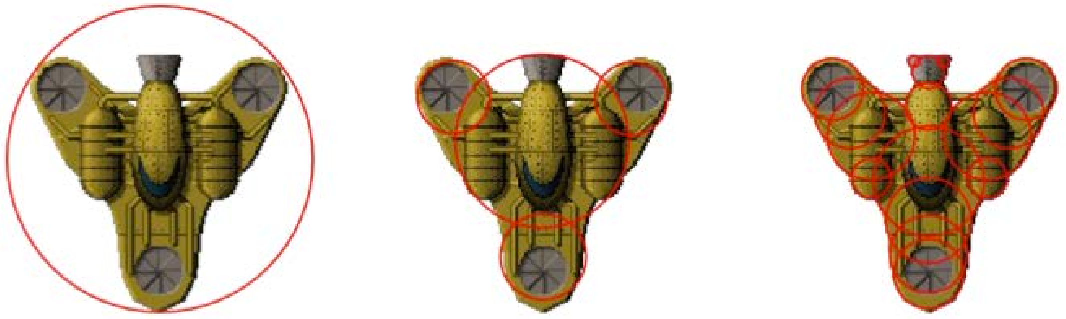
  - How do we know what direction the cross product should point in?
    - Use a reference point: e.g. point C
  - Test p against AB with C, BC with A, and CA with B

```
function SameSide(p1,p2, a,b)
    cp1 = CrossProduct(b-a, p1-a)
    cp2 = CrossProduct(b-a, p2-a)
    if DotProduct(cp1, cp2) >= 0 then return true
    else return false

function PointInTriangle(p, a,b,c)
    if SameSide(p,a, b,c) and SameSide(p,b, a,c)
        and SameSide(p,c, a,b) then return true
    else return false
```

Bounding Volume

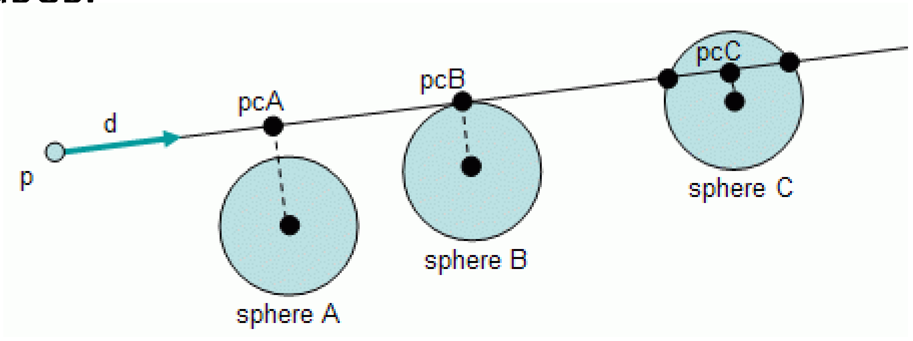
- A closed volume that completely contains the union of the objects in the set
  - Types(2D): rectangle, circle, ellipse, etc.
  - Types(3D): cuboid, sphere, capsule, cylinder, ellipsoid, etc.
- Bounding volume types
  - Sphere
  - Axis-aligned bounding box (AABB)
  - Object bounding box (OBB)





Bounding Volume

- Picking: ray-sphere intersection test (quick way)
  - 3 possible cases:



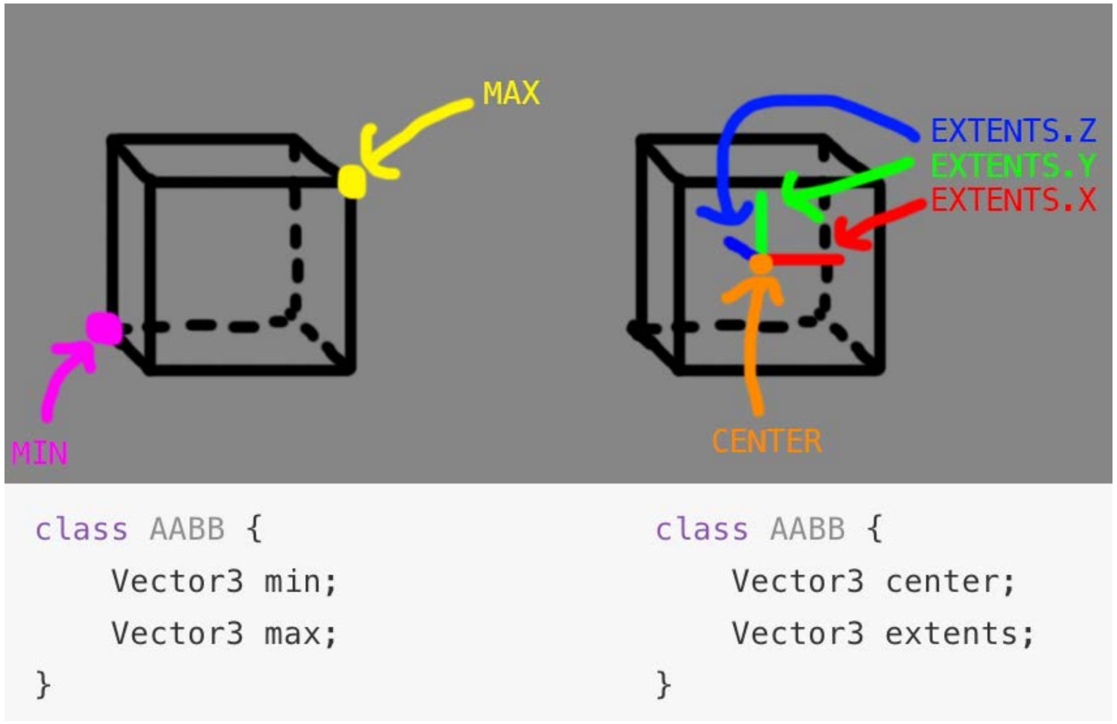
  - Find the closest point on the ray to the center of the object:

$$\mathrm{pc} = \mathrm{p} + \frac{(\mathrm{o} - \mathrm{p}) \cdot \mathrm{d}}{\mathrm{d} \cdot \mathrm{d}} \mathrm{d}$$

    - p: the position (or origin) of the pick ray
    - d: the direction of the pick ray
    - o: the center of the object

Bounding Volume

- Picking: ray-bounding box intersection test
  - Defining a bounding box model
  - Perform a ray-plane intersection test for the bounding box model



```
class AABB {
    Vector3 min;
    Vector3 max;
}
```

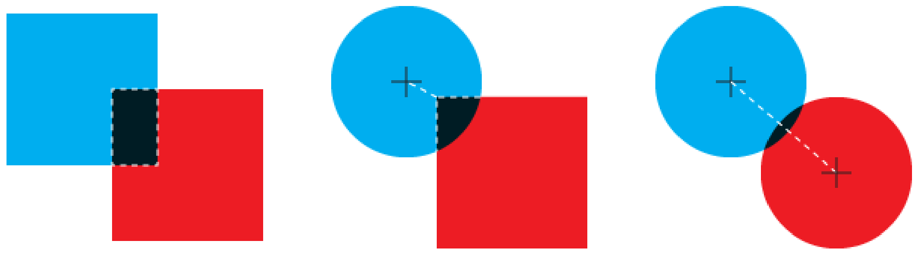
```
class AABB {
    Vector3 center;
    Vector3 extents;
}
```

Collision Detection

- Computational problem of detecting the intersection of two or more objects (in bounding volumes)
- Classification
  - Type of objects: rigid/soft, static/dynamic, 2D/3D, polygons/curves, convexity/concavity, etc.
  - Motion of objects: linear/nonlinear/angular, predictable/dynamic, bounded/unbounded, etc.
  - Time of collision: discrete/continuous, static/dynamic, exact/approximate, etc.
  - Other factors: real-time/off-line, number of objects, allowable error margin, frame vs path, etc.
- Components of real-time physics engine
  - Commercial: Havok, PhysX, Vortex, Rubikon, etc.
  - Open: Open Dynamics Engine (ODE), Bullet, SOFA, Vega, etc.

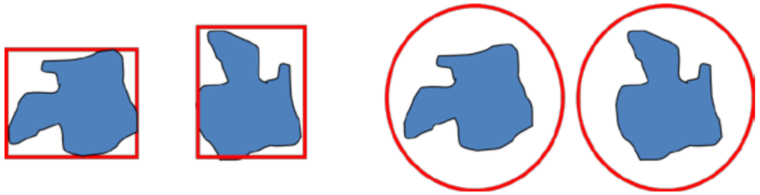
Collision Detection

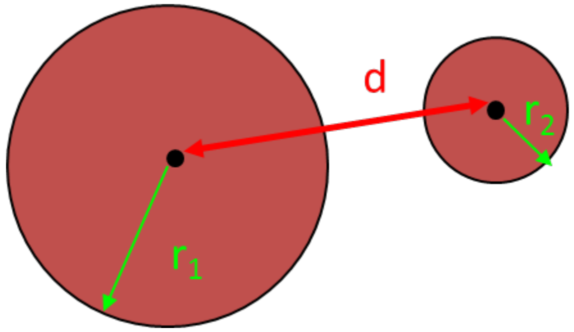
- Collision detection in video games
  - Limited computing time between several tasks or subsystems
  - Approximation of actual physics
    - Use simple rules or simplified models
  - 2D: overlapping pixels between sprites on the screen
  - 3D: overlapping polygons between meshes in the 3D space



Collision Detection

- Sphere
  - Sphere의 중심점을 기준으로 거리 계산
  - 빠른 계산 가능하고, object의 회전에 영향을 안받음
  - Object의 형태에 따라 부정확함

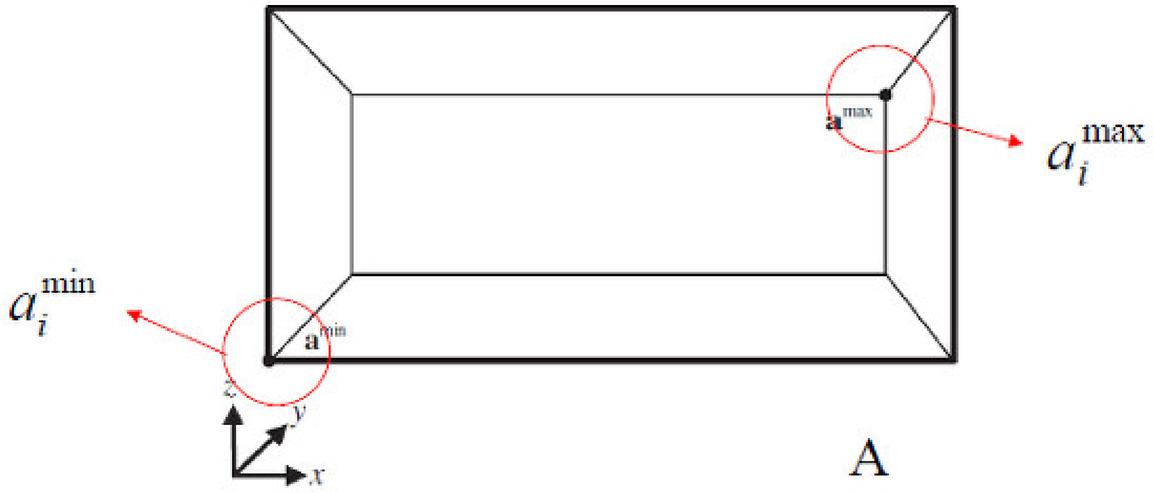


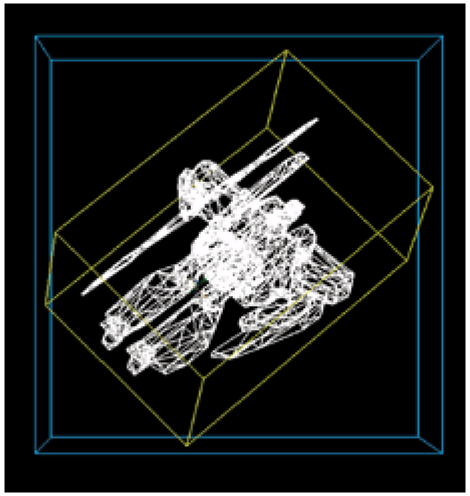


```
INT CollisionSphereToSphere(const D3DXVECTOR3* SphereCenter1
                            , FLOAT sphereRadius1
                            , const D3DXVECTOR3* SphereCenter2
                            , FLOAT sphereRadius2 )
{
        INT hr=-1;

        FLOAT fDistance;
        D3DXVECTOR3 vcTemp = *SphereCenter1 - *SphereCenter2;

        fDistance = D3DXVec3Length(&vcTemp);

        if(fDistance <= (sphereRadius1 + sphereRadius2))
                hr =0;

        return hr;
}
```

Collision Detection

- Axis-aligned bounding box (AABB)
  - Box를 이루는 면의 normal vector들이 x, y, z axis와 일치
  - Model을 이루는 다각형의 x, y, z 좌표의 최소, 최대를 각 box의 vertex로 생성 → Object의 회전시 크기가 계속 변함

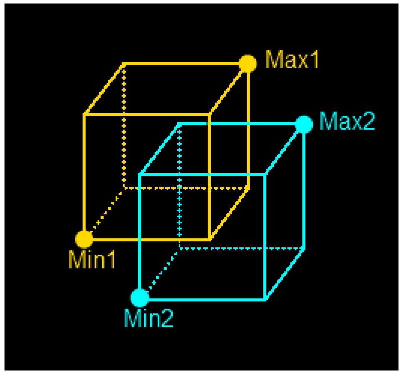


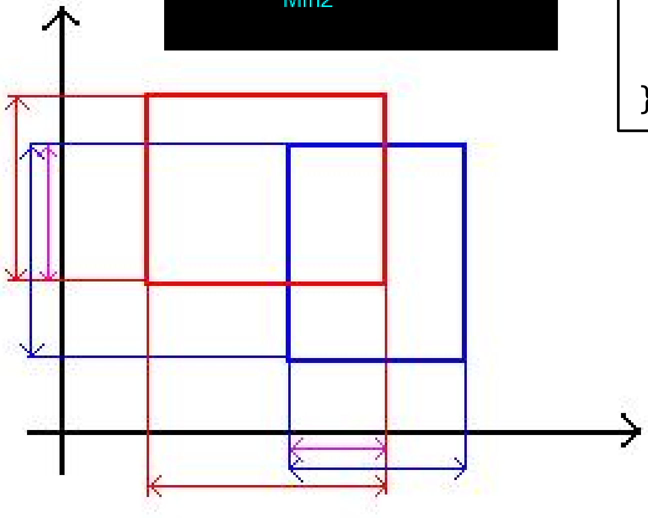


AABB called A is described by two extreme points call a_i^{min} and a_i^{max}, where $a_i^{min} \le a_i^{max}, \forall i \in \{x, y, z\}$.

Collision Detection

- Axis-aligned bounding box (AABB)
  - 축에 일치된 모양이기 때문에 AABB끼리의 충돌 검출이 매우 간편
  - 회전하는 object의 경우 정확한 충돌 검사가 어려움

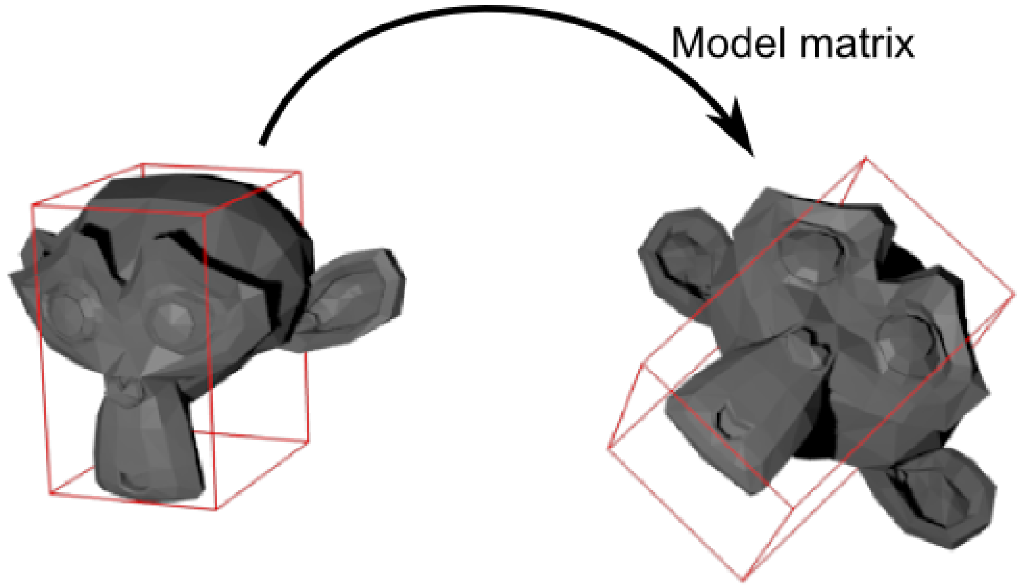


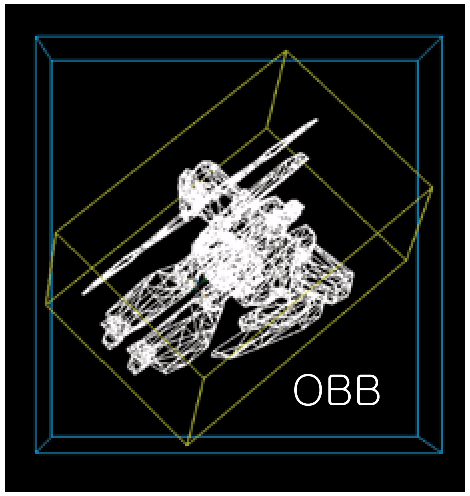


```
void CheckAABB(BoundingBox targetBox) {
	if  (boundingBox.minPos[0] <= targetBox.maxPos[0] &&
	     boundingBox.maxPos[0] >= targetBox.minPos[0] &&
	     boundingBox.minPos[1] <= targetBox.maxPos[1] &&
	     boundingBox.maxPos[1] >= targetBox.minPos[1] &&
	     boundingBox.minPos[2] <= targetBox.maxPos[2] &&
	     boundingBox.maxPos[2] >= targetBox.minPos[2]) {
	     aabbCollide = true;
	     return;
	}

	aabbCollide = false;
}
```

Collision Detection

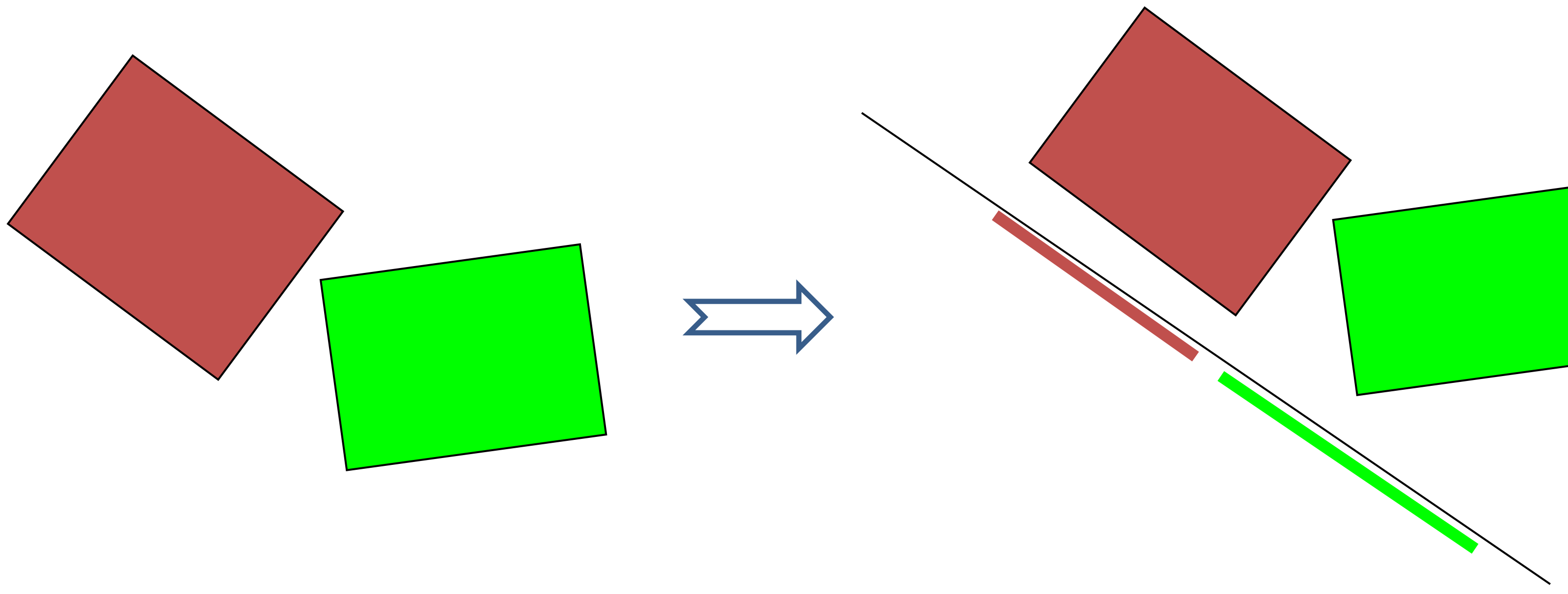
- Object bounding box (OBB)
  - AABB처럼 box를 이루는 면은 서로 수직이지만, 해당 면의 normal vector가 x, y, z axis와 일치하지 않음
  - Mesh를 transform 할 때 box도 같이 transform 적용
    - Object가 회전할 때마다 box의 범위를 다시 계산할 필요가 없고, AABB보다 세밀한 충돌 검사 가능
    - x, y, z axis와 일치하지 않기 때문에 AABB보다 충돌 검사의 시간 및 복잡도가 높음





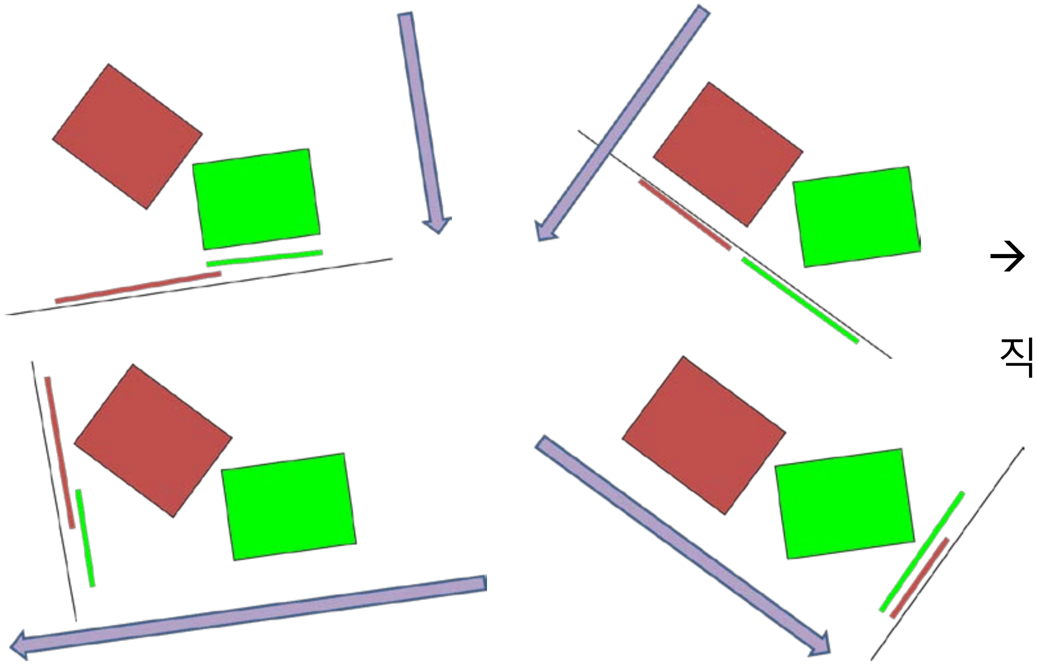
Collision Detection

- Object bounding box (OBB)
  - OBB끼리의 충돌 검사: separating axis theory 사용
    - 임의의 두 block 다면체 A, B에 대해 어떤 축이 존재해서 그 축(separating axis)으로의 다면체들의 투영들의 구간이 서로 겹치지 않는다면 A, B는 서로 분리돼 있음
    - 즉, 두 block 다면체에 대해서 분리축이 존재하면 두 다면체가 충돌하지 않음



Collision Detection

- Object bounding box (OBB)
  - OBB끼리의 충돌 검사: separating axis theory 사용
    - Separating axis은 다음의 axis들 중 하나에 평행한 axis으로 결정
      1. A의 면 중, 하나의 면 법선 벡터
      2. B의 면 중, 하나의 면 법선 벡터
      3. A의 변과 B의 변에 동시에 수직인 축 (3D object의 경우)

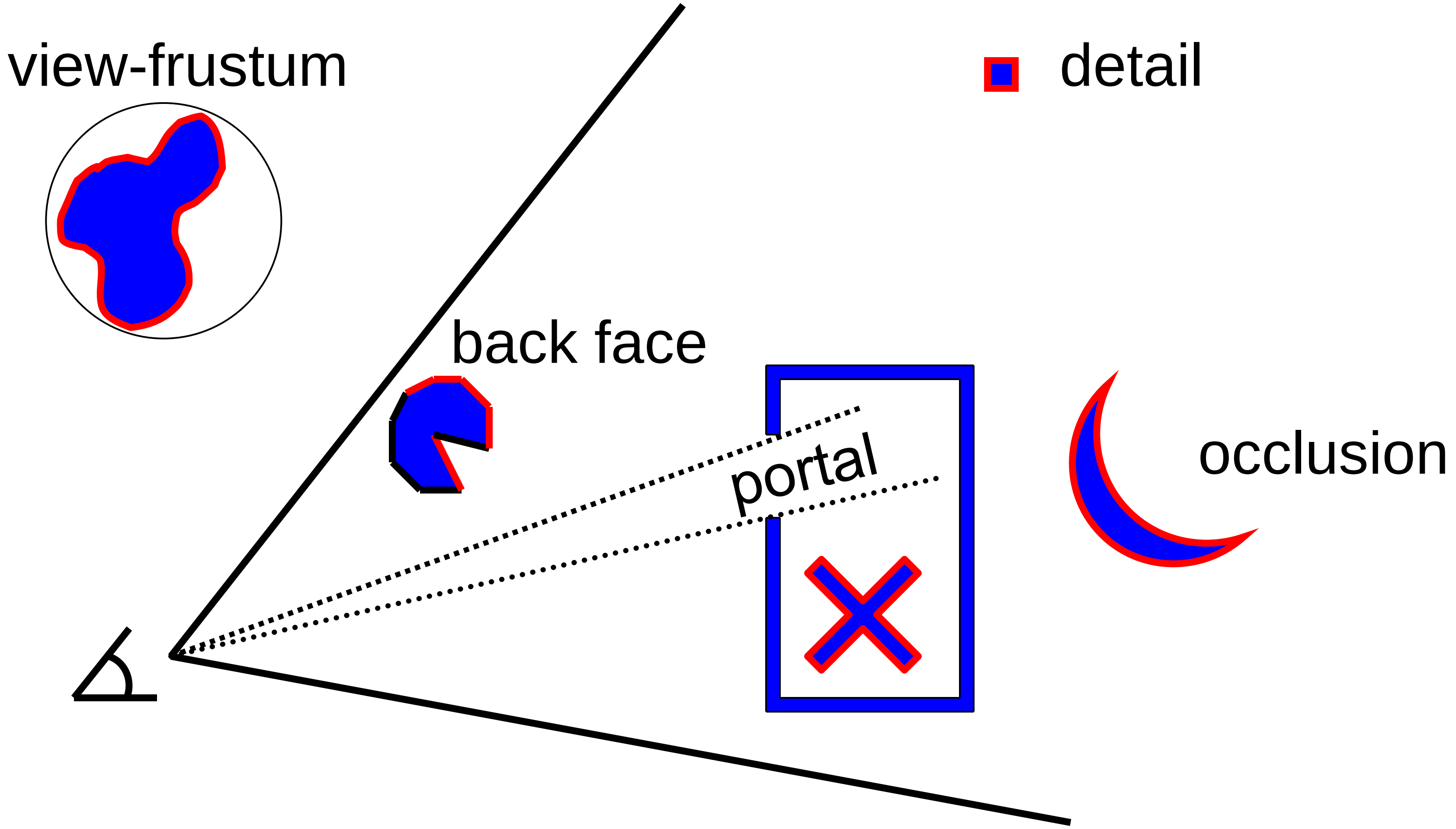


A의 **n** = 6, edge =12
B의 **n** = 6, edge = 12
→ Total 분리축 후보: 12 x 12 + 6 + 6 = 156

직육면체이기 때문에 서로 평행한 축 제외시
면 **n** = 3, edge = 3
→ 실제 검사: 3 x 3 + 3 + 3 = 15

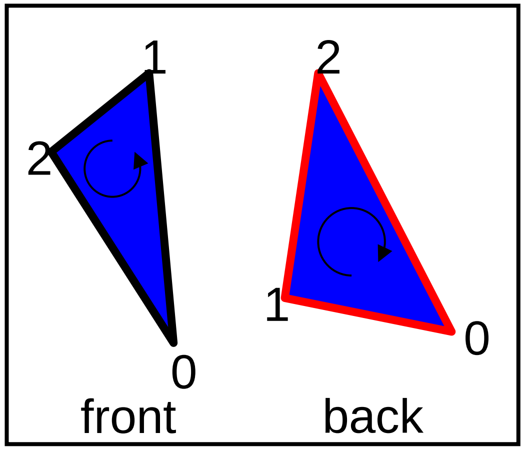
Visibility Culling

- Culling examples

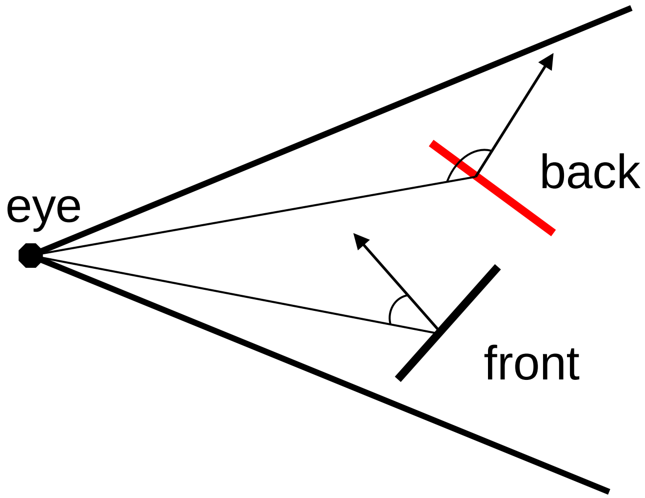


Visibility Culling

- Back Face Culling
  - Discard polygons that faces away from the viewer
  - Can be used for
    - closed surface (e.g. sphere)
    - the back faces never should be seen (e.g. walls in a room)
  - Direct3D: D3D11_CULL_BACK
  - Two methods:
    - Screen space
    - Eye space



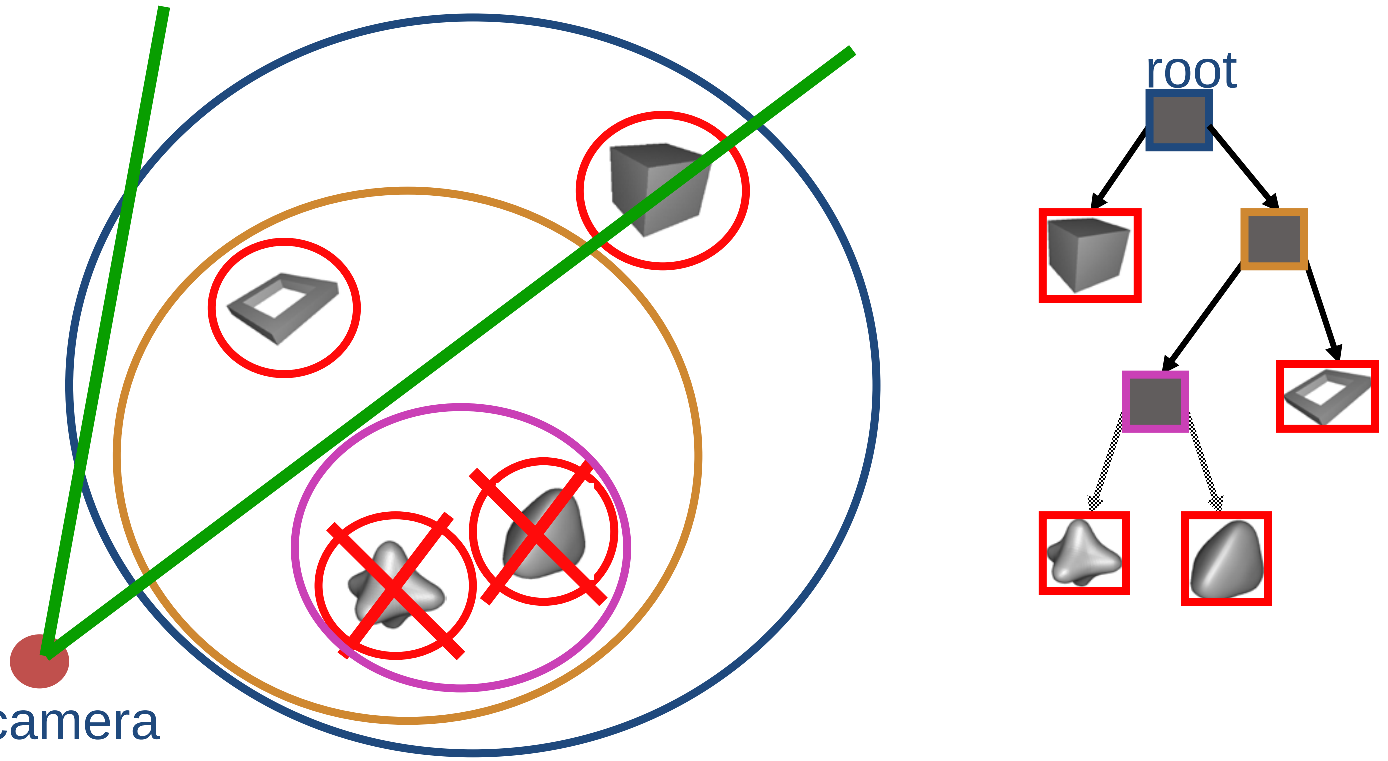
screen space



eye space

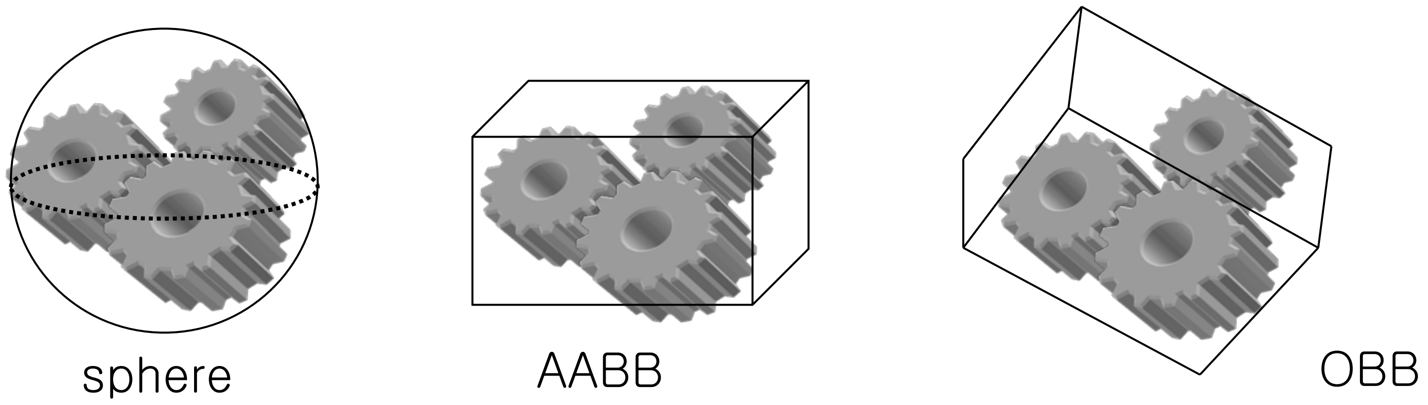
Visibility Culling

- View-Frustum Culling
  - Bound every group of primitives by a simple volume (e.g. sphere, cube, etc…)
  - How to accelerate view-frustum culling further?
    - Use a hierarchical structure (e.g. the scene graph)



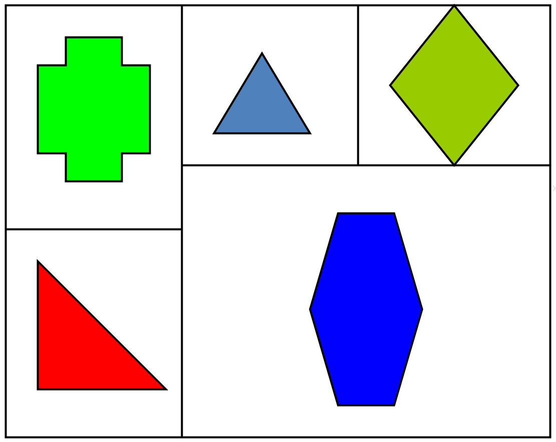
Visibility Culling

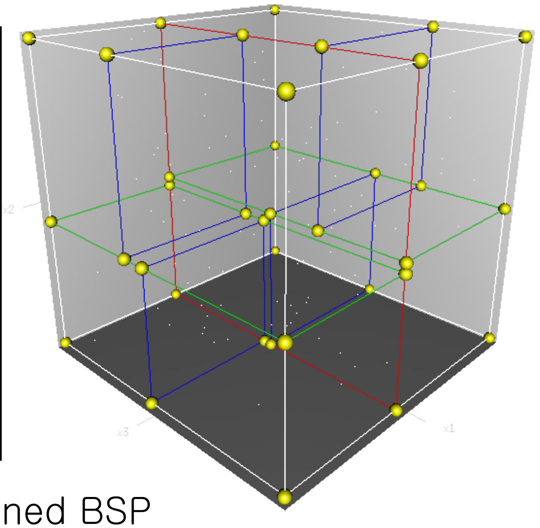
- View-Frustum Culling
  - Culling against bounding volumes to save time
  - Easy to compute & as tight as possible
  - Bounding volumes (boxes)
    - Sphere, cube, etc…
    - Axis-aligned bounding box (AABB)
    - Object bounding box(OBB)



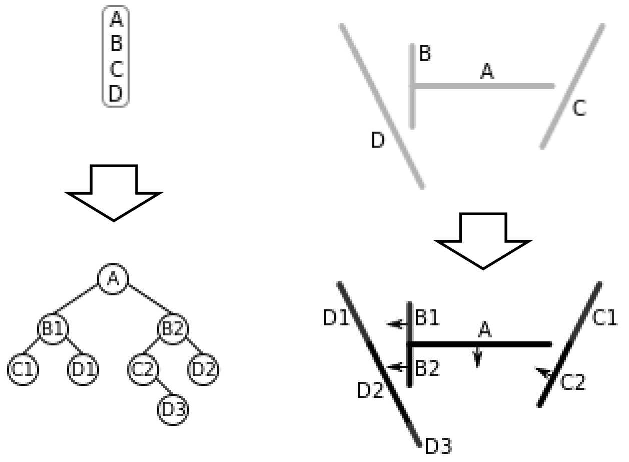
Visibility Culling

- View-Frustum Culling: Binary space partitioning (BSP) tree
  - A method of recursively subdividing a space (scene) into two until the partitioning satisfies one or more requirements
  - This subdivision gives rise to a representation of objects within the space by means of a tree data





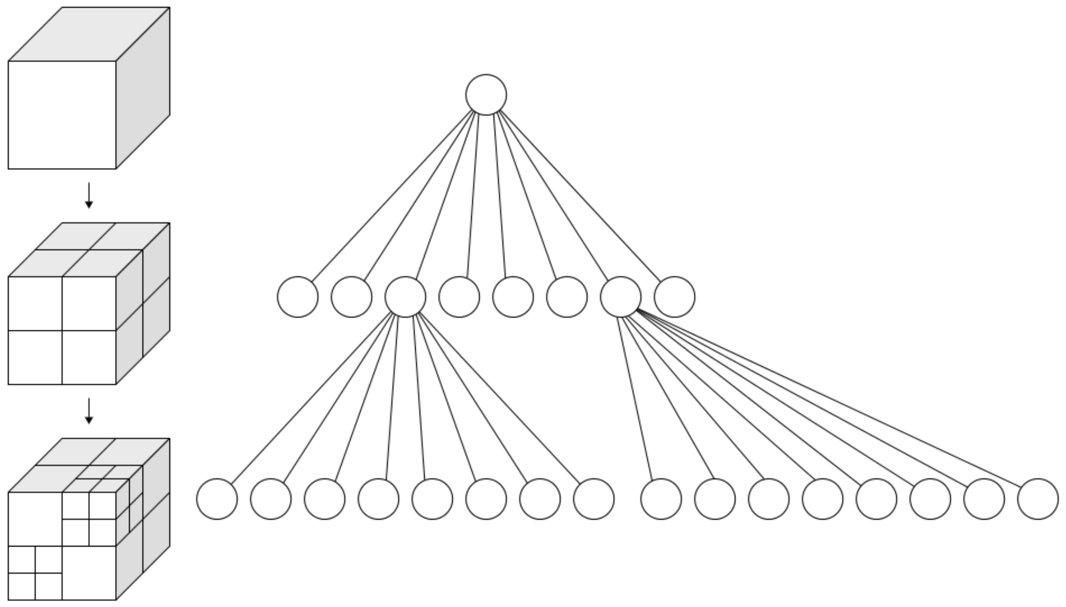
axis−aligned BSP
(e.g. *k*−d tree)



polygon−aligned BSP

Visibility Culling

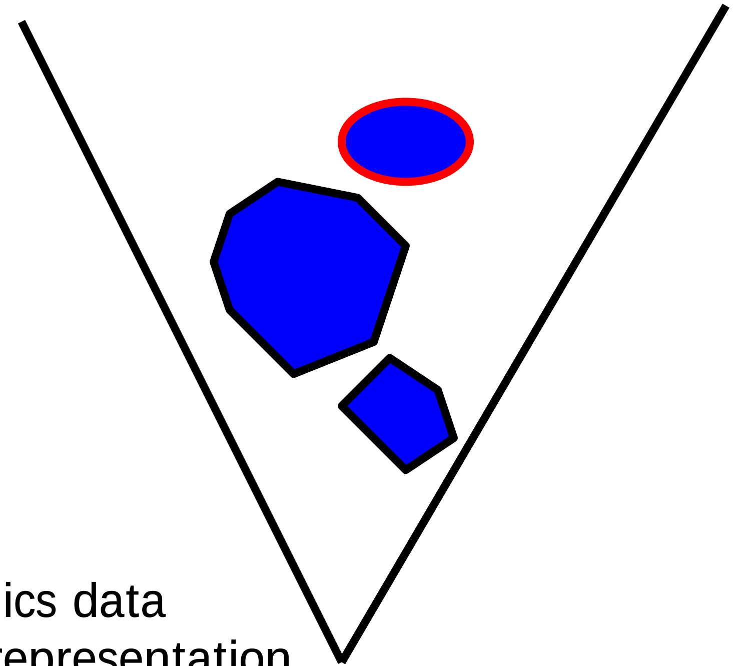
- View-Frustum Culling: Octree
  - Each parent has exactly eight children
  - The point defines one of the corners for each of the eight children
  - Subdivide the space until the number of primitives within each leaf node is less than a threshold (e.g. number of points)
  - Can be used with depth-first search(DFS) to render only surfaces



Visibility Culling

- Occlusion Culling
  - Discard objects that lies completely behind another set of objects
  - Occlusion culling algorithm:

```
OcclusionCulling (G)
Or = empty
For each object g in G
    if (isOccluded(g, Or))
           skip g
    else
           render (g)
         update (Or)
    end
End
```

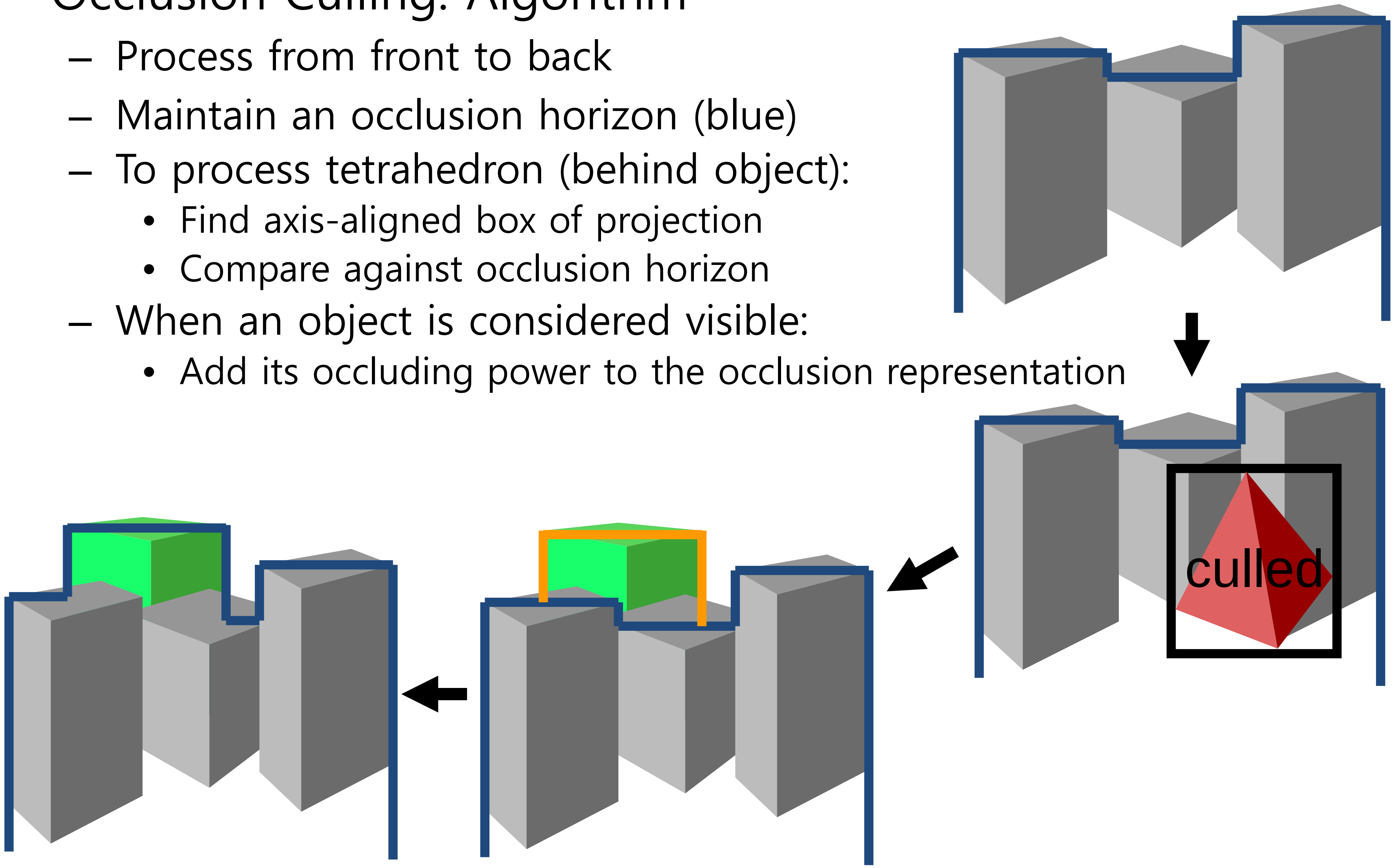


G: input graphics data
Or: occlusion representation

The problem:
1. algorithms for isOccluded()
2. Fast update Or

Visibility Culling

- Occlusion Culling: Algorithm
  - Process from front to back
  - Maintain an occlusion horizon (blue)
  - To process tetrahedron (behind object):
    - Find axis-aligned box of projection
    - Compare against occlusion horizon
  - When an object is considered visible:
    - Add its occluding power to the occlusion representation

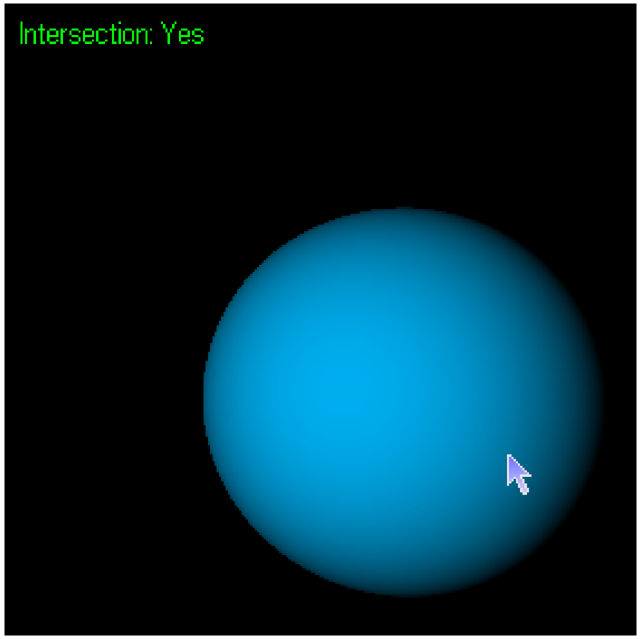


Tutorials

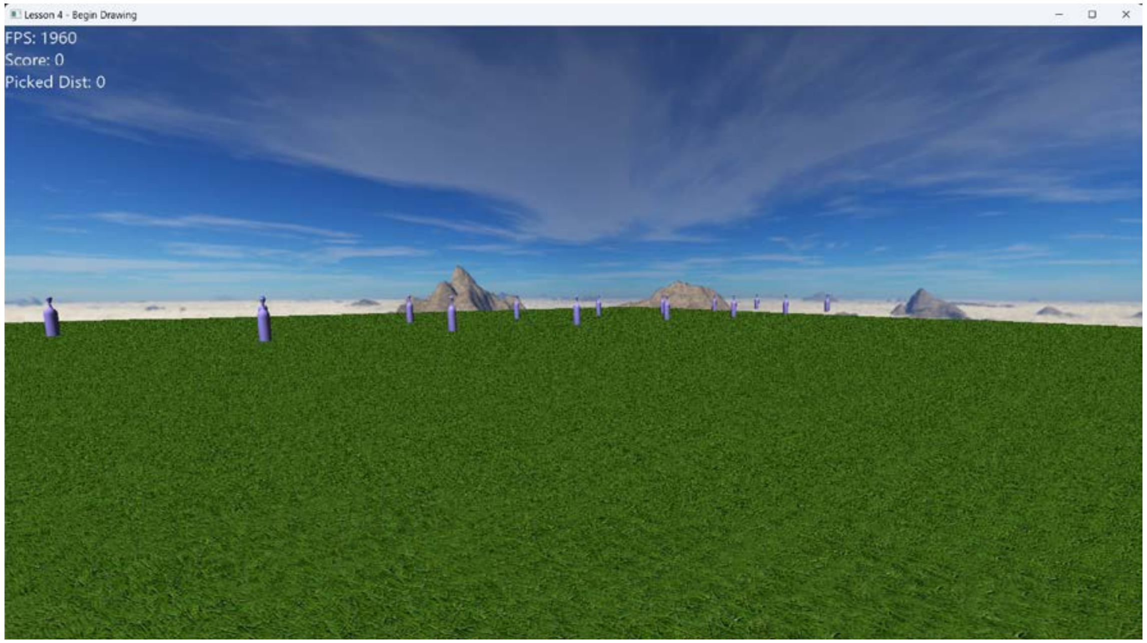
- Picking
- Bounding Volumes
- Collision Detection
- View-Frustum Culling

8-1 Picking

- Mouse picking samples (*RasterTek, BraynzarSoft)
  1. Get the mouse cursor position in screen space
  2. Transform 2D screen space ray to 3D view space ray
  3. View space to world space
  4. Transform vertices to world space
  5. Check for an intersection between the pick ray and object model



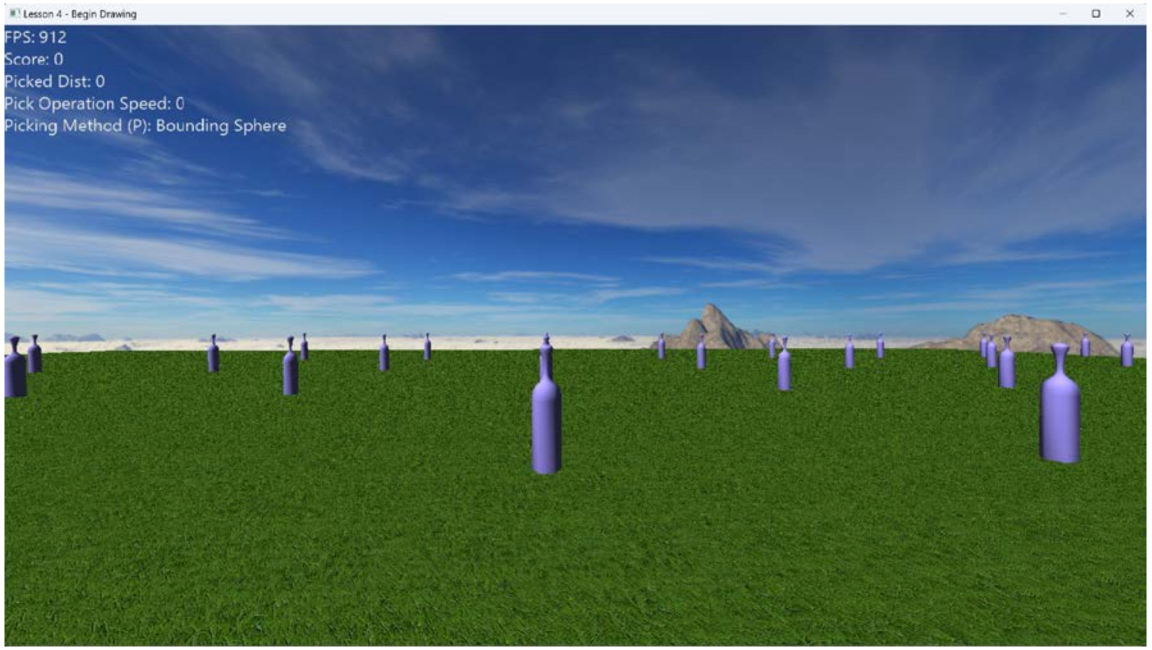
Intersection: Yes



FPS: 1960
Score: 0
Picked Dist: 0

8-2 Bounding Volume

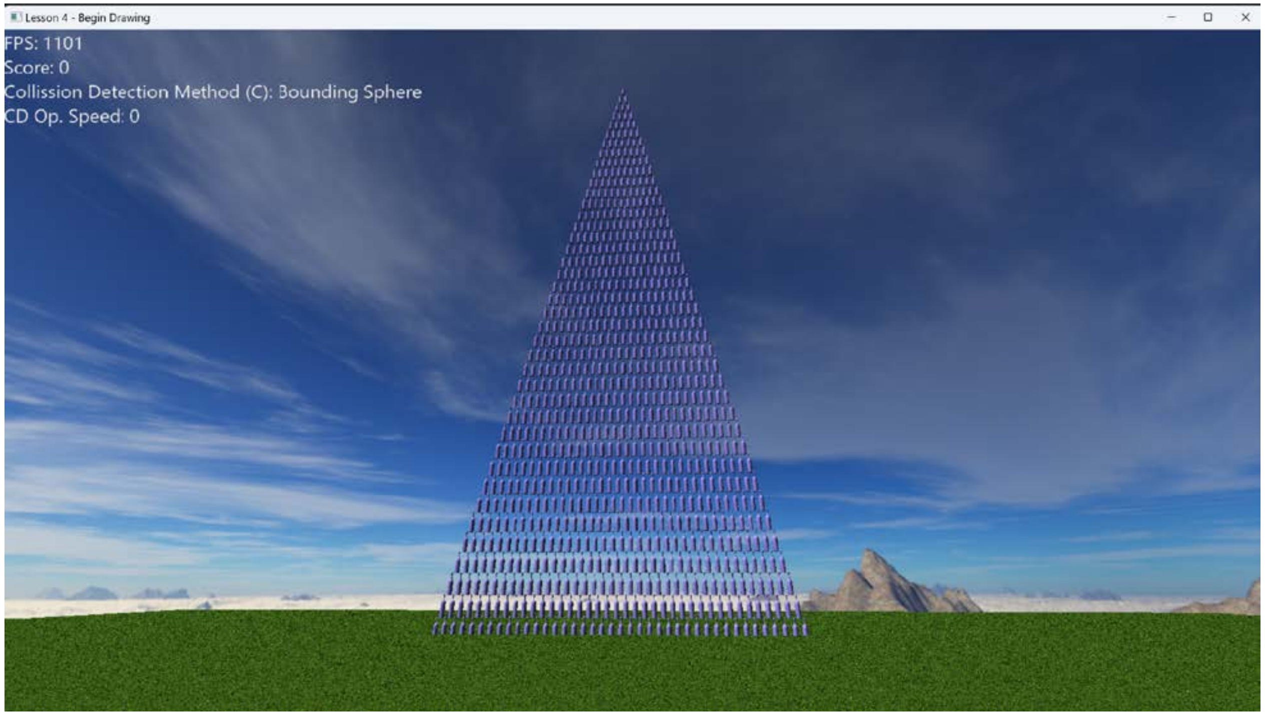
- Bounding volumne sample (*BraynzarSoft)
  - Testing with bounding box, sphere, and mesh
    - Use “p” key to change the different bounding volume
  - Relying on the picking ray of the previous tutorial



```
FPS: 912
Score: 0
Picked Dist: 0
Pick Operation Speed: 0
Picking Method (P): Bounding Sphere
```

8-3 Collision Detection

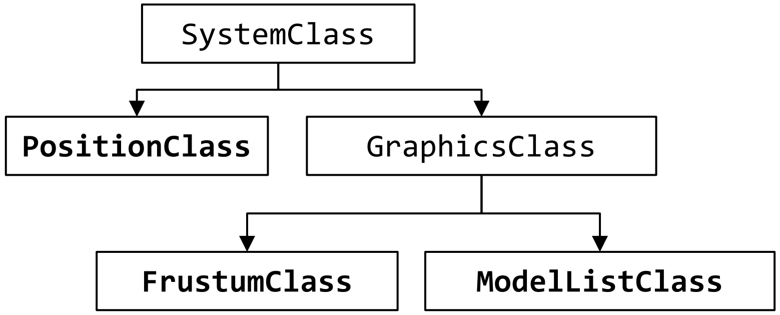
- Collision detection sample (*BraynzarSoft)
  - Testing with bounding box and sphere
    - Use “c” key to change the different bounding volume
  - Relying on the collision detection of the previous tutorial

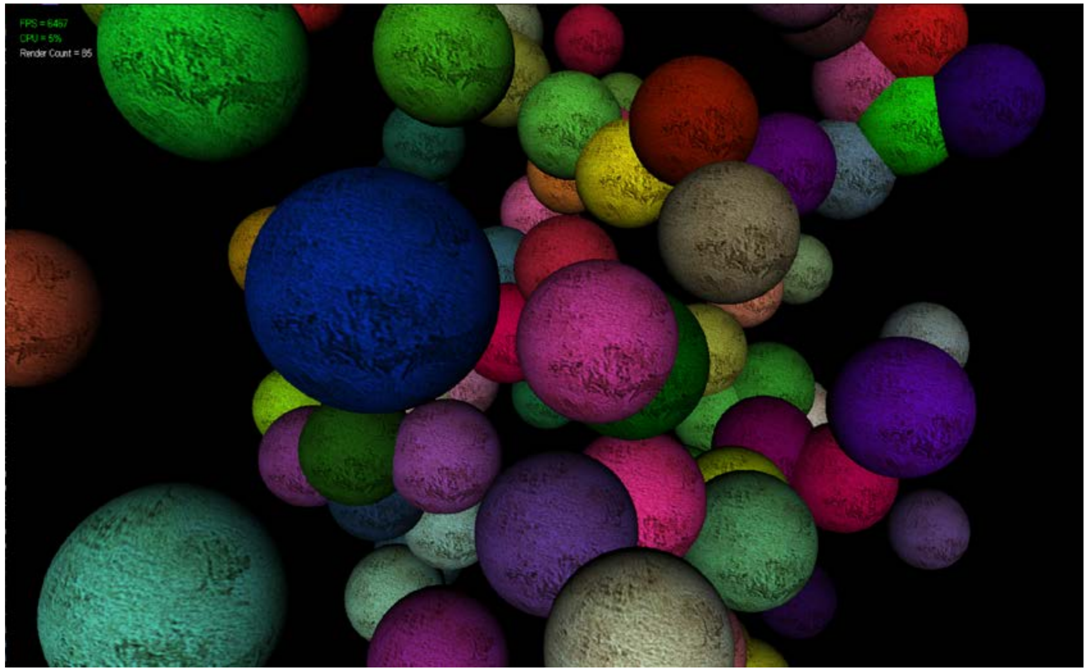


FPS: 1101
Score: 0
Collission Detection Method (C): Bounding Sphere
CD Op. Speed: 0

8-4 View-Frustum Culling

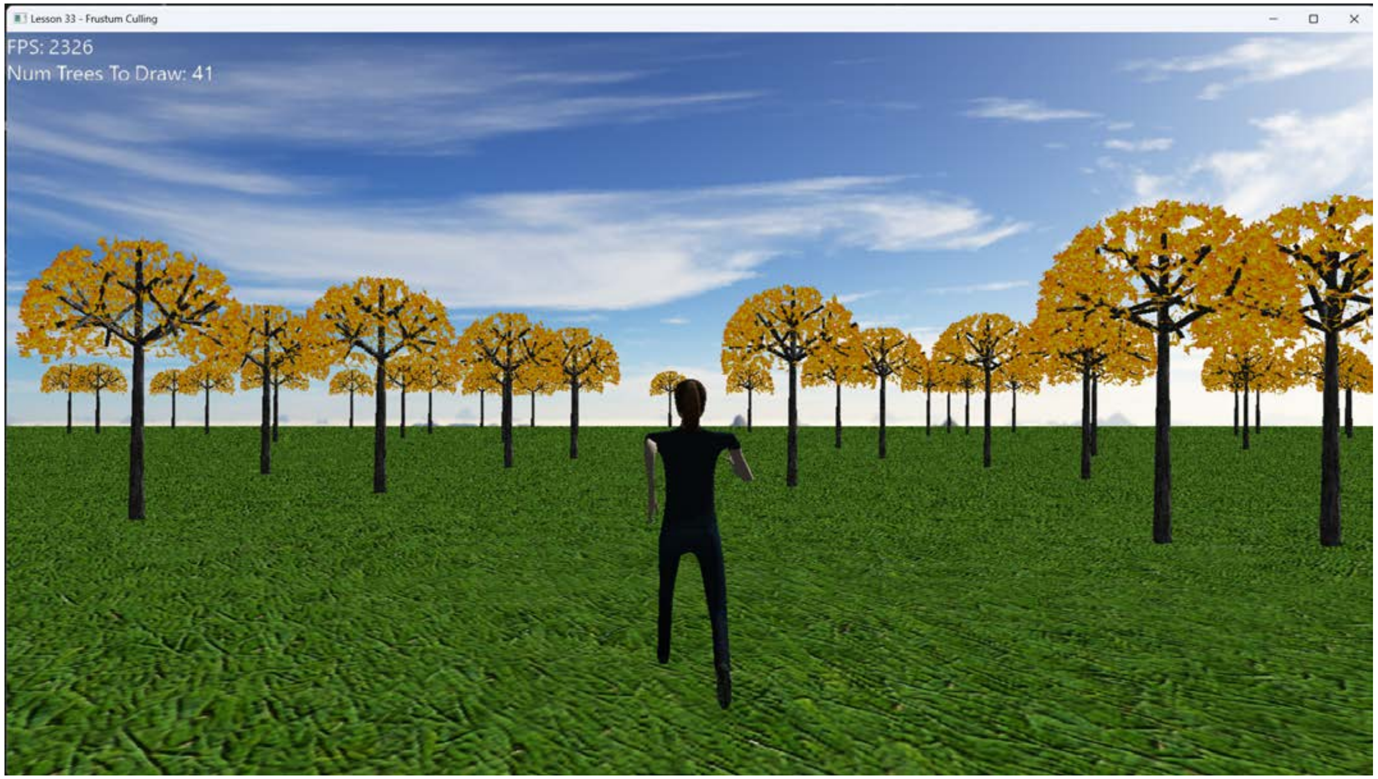
- Adding view-frustum culling to the framework (*RasterTek)
  - PositionClass: handles left/right camera movement
  - FrustumClass: uses the six planes of the view frustum based on the updated viewing location and checks the object inside the frustum
  - ModelListClass: maintains information(the size and color of the sphere) about all the models in the scene





8-4 View-Frustum Culling

- View-frustum culling with AABB (*BraynzarSoft)
  - 4000 trees in the scene
  - Check an object with AABB within the camera’s view
    - The test is done with CPU before sending to GPU
    - GPU does frustum culling with every triangle



FPS: 2326
Num Trees To Draw: 41

References

- Wikipedia
  - www.wikipedia.org
- Introduction to DirectX 11
  - www.3dgep.com/introduction-to-directx-11
- Raster Tek
  - www.ratertek.com
- Braynzar Soft
  - www.braynzarsoft.net)
- CS 445: Introduction to Computer Graphics *[Aaron Bloomield]*
  - www.cs.virginia.edu/~asb/teaching/cs445-fall06

Q & A