Introduction to Graphics Programming and its Applications

繪圖程式設計與應用

Mesh Rendering

Instructor: Hung-Kuo Chu

Department of Computer Science

National Tsing Hua University

CS4595

Codeblock Conventions

Yellow Codeblock => Application Program (CPU)

- Create OpenGL Context
- Create and Maintain OpenGL Objects
- Generate Works for the GPU to Consume

Blue Codeblock => Shader Program (GPU)

- Shader for a Certain Programmable Stage
- Process Geometry or Fragment in Parallel
- Starts with #version 410 core Declaration

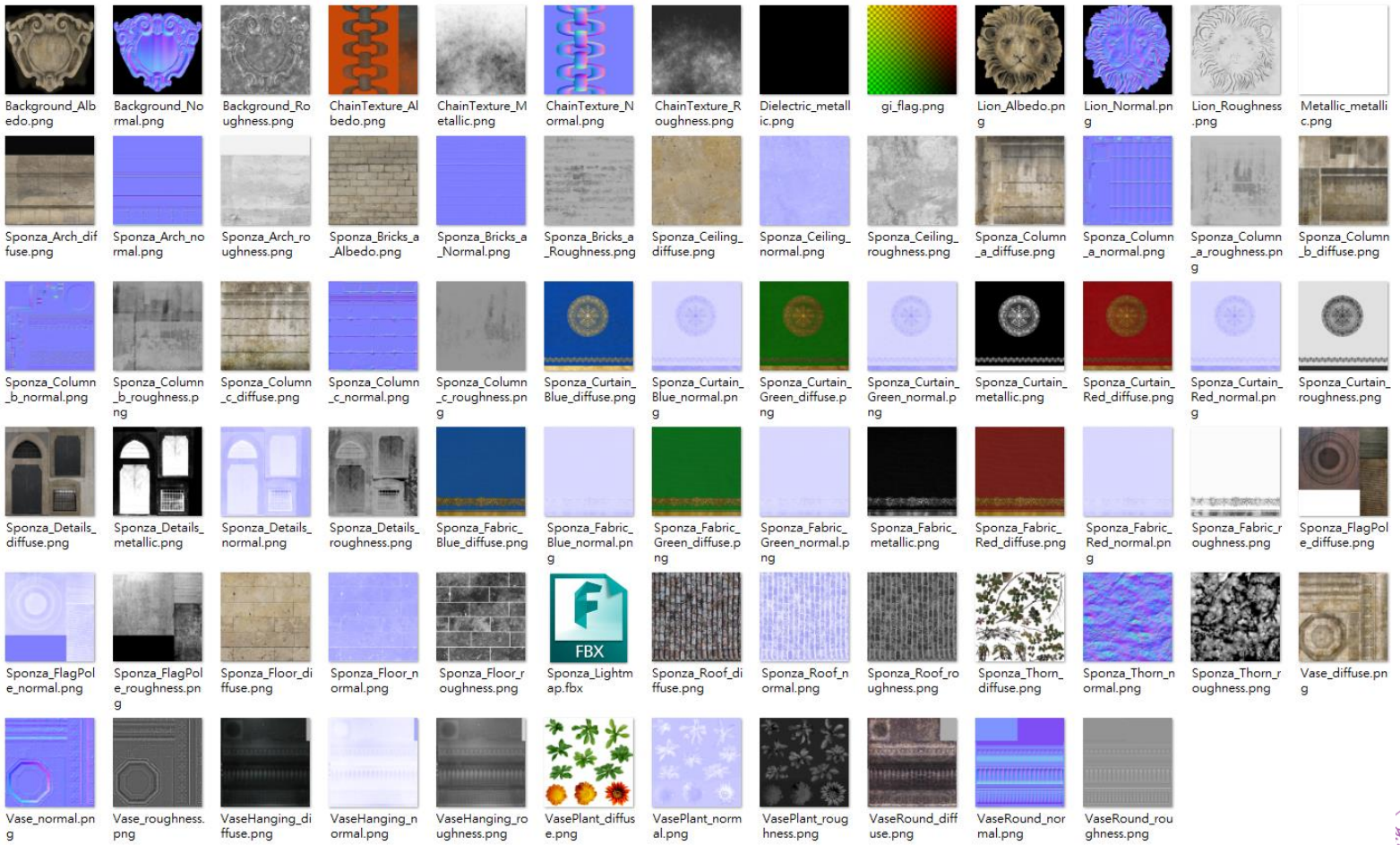
What You'll Learn in This Lecture

- Properly process an input scene model
- Render a scene using OpenGL
- Accelerating scene rendering
- Vertex skinning (mesh animation)

MESH RENDERING

Mesh Rendering: Goal

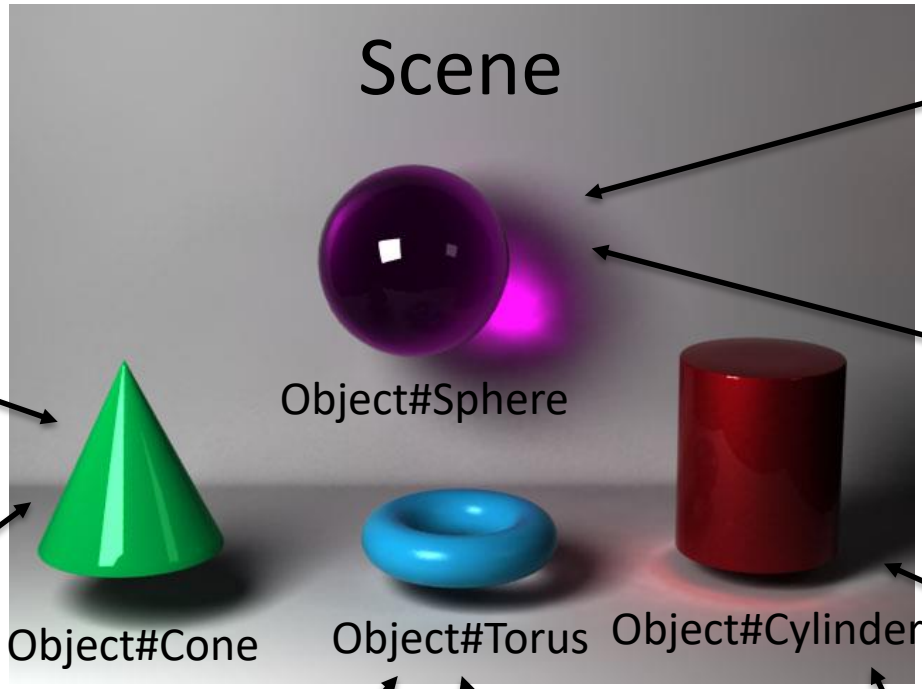
…From These Files

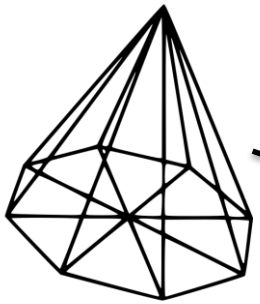


3D Model

- Work by 3D modeler artists
- Defines a scene consisting of shapes, materials and other attributes
  - Scene: layout/arrangement of objects
  - Shapes: vertices and indices of surfaces (triangles!)
  - Materials: lighting property of a surface

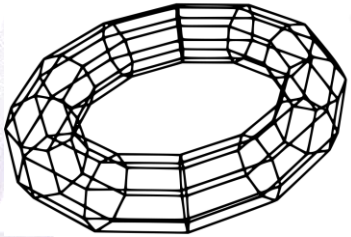
3D Model





Shape#Cone

Material#Green



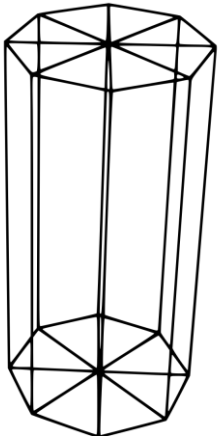
Shape#Torus

Material#Cyan

Material#Red

Shape#Sphere

Material#Purple



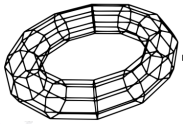
Shape#Cylinder

Rendering a Scene

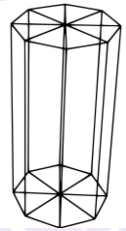
- Idea: render one object at a time
- Load each shape as a VAO, material as Texture


→ GLuint cone_vao;


→ GLuint cone_tex;


→ GLuint torus_vao;


→ GLuint torus_tex;


→ GLuint cylinder_vao;

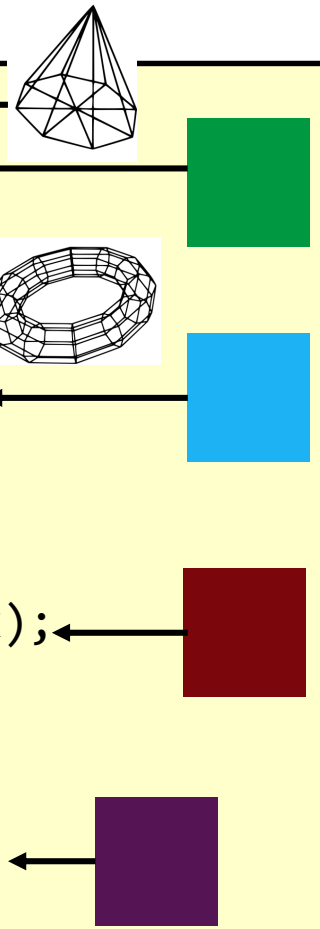

→ GLuint cylinder_tex;


→ GLuint sphere_vao;


→ GLuint sphere_tex;

Rendering a Scene

```
glBindVertexArray(cone_vao);
glBindTexture(GL_TEXTURE_2D, cone_tex);
glDrawElements(…);

glBindVertexArray(torus_vao);
glBindTexture(GL_TEXTURE_2D, torus_tex);
glDrawElements(…);

glBindVertexArray(cylinder_vao);
glBindTexture(GL_TEXTURE_2D, cylinder_tex);
glDrawElements(…);

glBindVertexArray(sphere_vao);
glBindTexture(GL_TEXTURE_2D, sphere_tex);
glDrawElements(…);
```



3D Model Formats

- Autodesk (.fbx)
- Collada (.dae)
- glTF (.gltf, .glb)
- Blender 3D (.blend)
- 3ds Max 3DS (.3ds)
- 3ds Max ASE (.ase)
- Wavefront Object (.obj)
- Industry Foundation Classes (IFC/Step) (.ifc)
- XGL (.xgl,.zgl)
- SDKMesh (.sdkmesh)
- Stanford Polygon Library (.ply)
- AutoCAD DXF (.dxf)
- LightWave (.lwo)
- LightWave Scene (.lws)
- Modo (.lxo)
- Stereolithography (.stl)
- DirectX X (.x)
- AC3D (.ac)
- Milkshape 3D (.ms3d)
- TrueSpace (.cob,.scn)

3D Model Formats

- Created to suit artist/rendering needs
- Each one has its own purpose and feature
- .OBJ and .FBX are the most common in the case of free 3D models
- What are encoded in these?

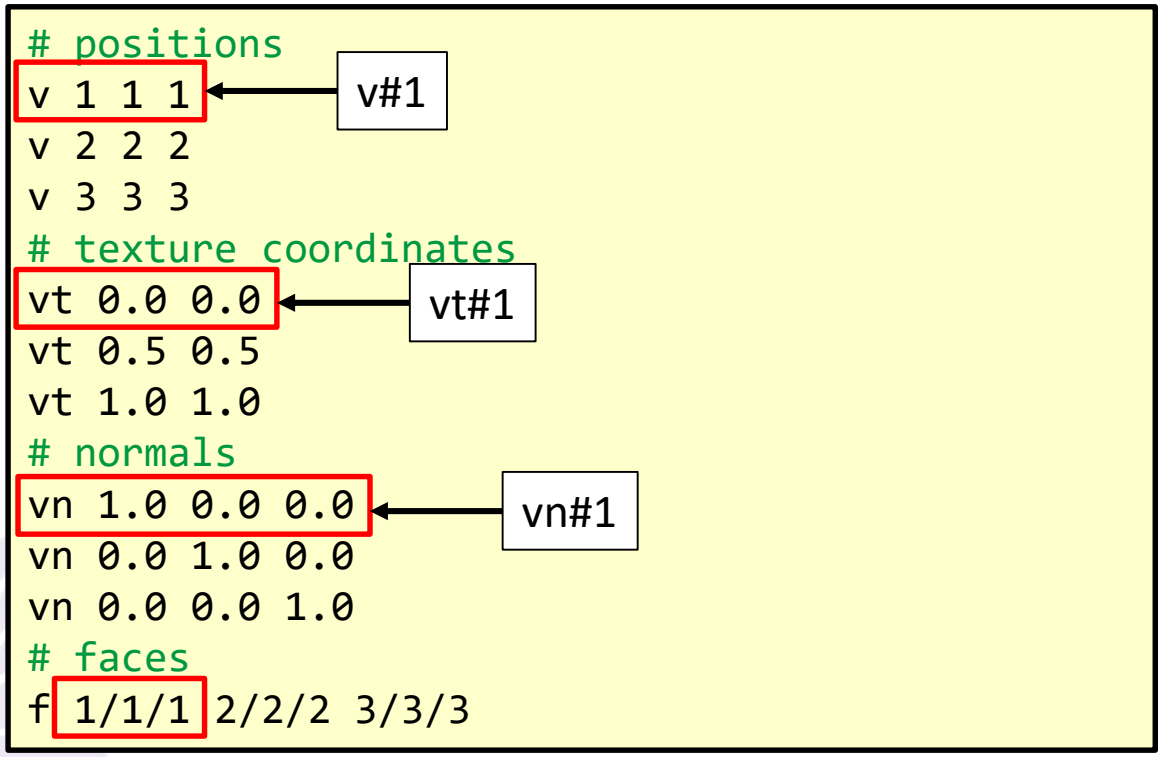
Wavefront OBJ

```
# List of geometric vertices in (x,y,z[,w])
  v 0.123 0.234 0.345 1.0
  v ...
  ...
# List of texture coordinates in (u, v [,w])
  vt 0.500 1 [0]
  vt ...
  ...
# List of vertex normals in (x,y,z)
  vn 0.707 0.000 0.707
  vn ...
  ...
# Polygonal face element
  f 1 2 3
  f 3/1 4/2 5/3
  f 6/4/1 3/5/3 7/6/5
  f 7//1 8//2 9//3
  f ...
  ...
```

Wavefront OBJ

- Faces are defined by multiple vertices

```
f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 ...
```



Wavefront OBJ

- Materials

```
newmtl Textured
   Ka 1.000 1.000 1.000        # Ka, Kd, Ks, d, illum are
   Kd 1.000 1.000 1.000        # Phong BRDF parameters
   Ks 0.000 0.000 0.000
   d 1.0
   illum 2
   map_Ka lemur.tga            # the ambient texture map
   map_Kd lemur.tga            # the diffuse texture map
   map_Ks lemur.tga            # specular color texture map
   map_Ns lemur_spec.tga       # specular highlight component
   map_d lemur_alpha.tga       # the alpha texture map
   map_bump lemur_bump.tga     # bump(normal) map
```

Wavefront OBJ

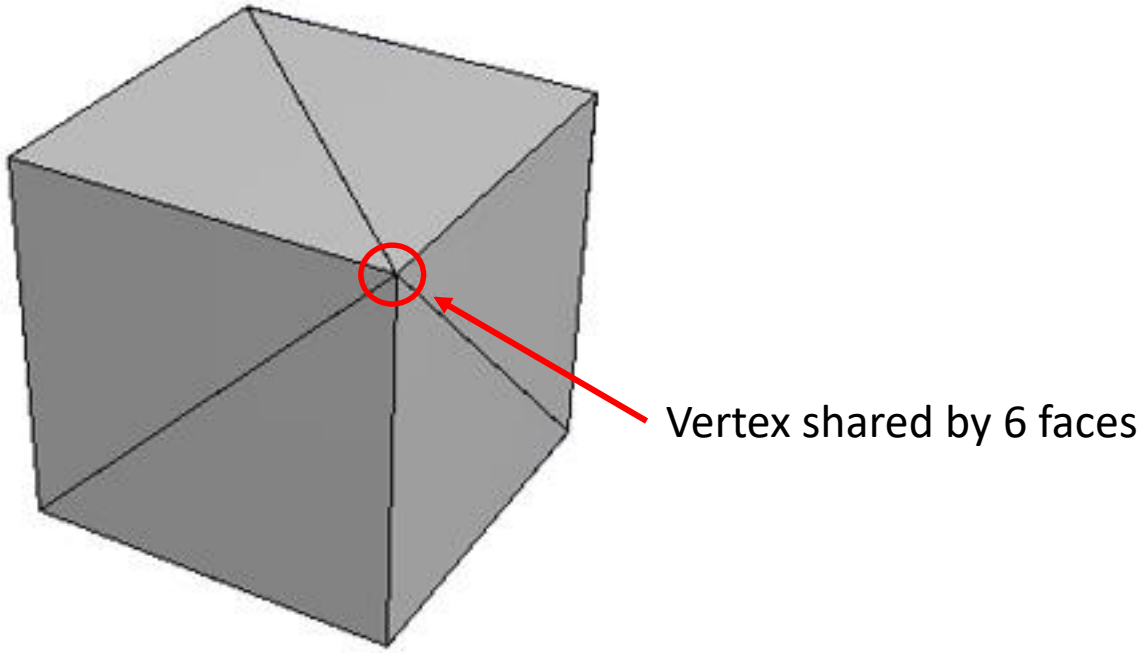
- Materials

```
usemtl Mat1
# following triangles uses Mat1
f 1/1/1 2/2/2 3/3/3
f 4/4/4 5/5/5 6/6/6

usemtl Mat2
#following triangles uses Mat2
f 7/7/7 8/8/8 9/9/9
f 10/10/10 11/11/11 12/12/12
```

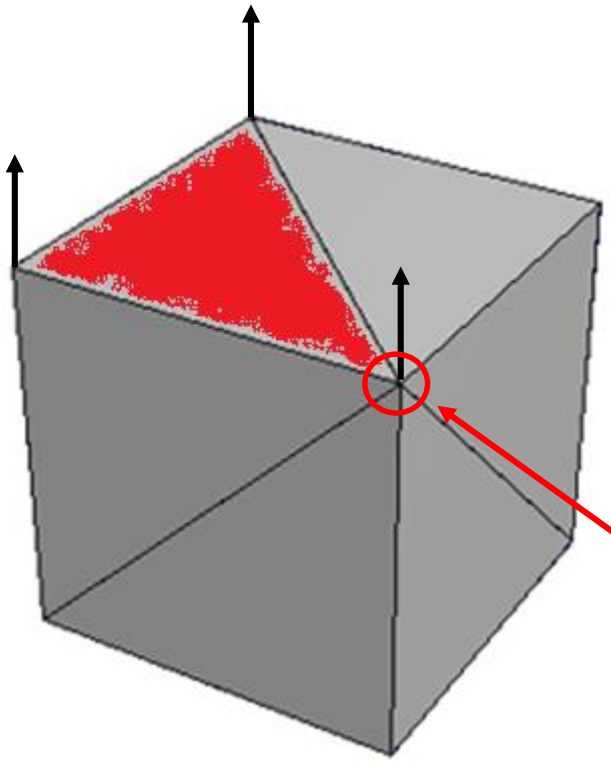
3D Model Format Problems

- Missing normal
  - Need to compute it from positions
- Material texture maps given in absolute path
  - Need to fix the file path
- Per-face/per-vertex normal
  - Need to split vertex

Per-face/per-vertex normal

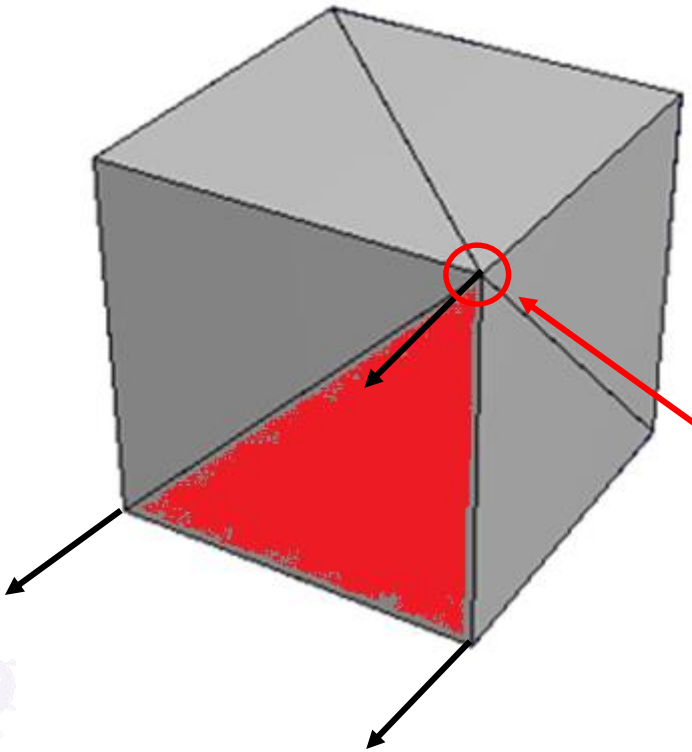


Per-face normal



Per-face normal

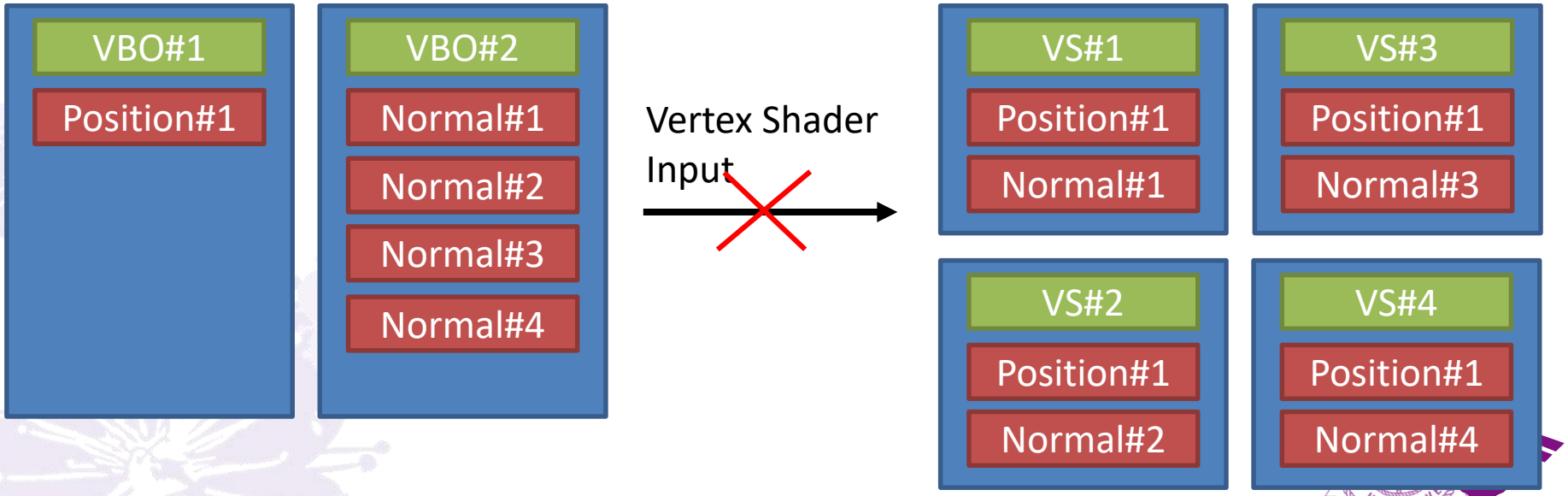
Conclusion: each vertex should be represented by 1 position + 6 normal



While it should be forward for this face

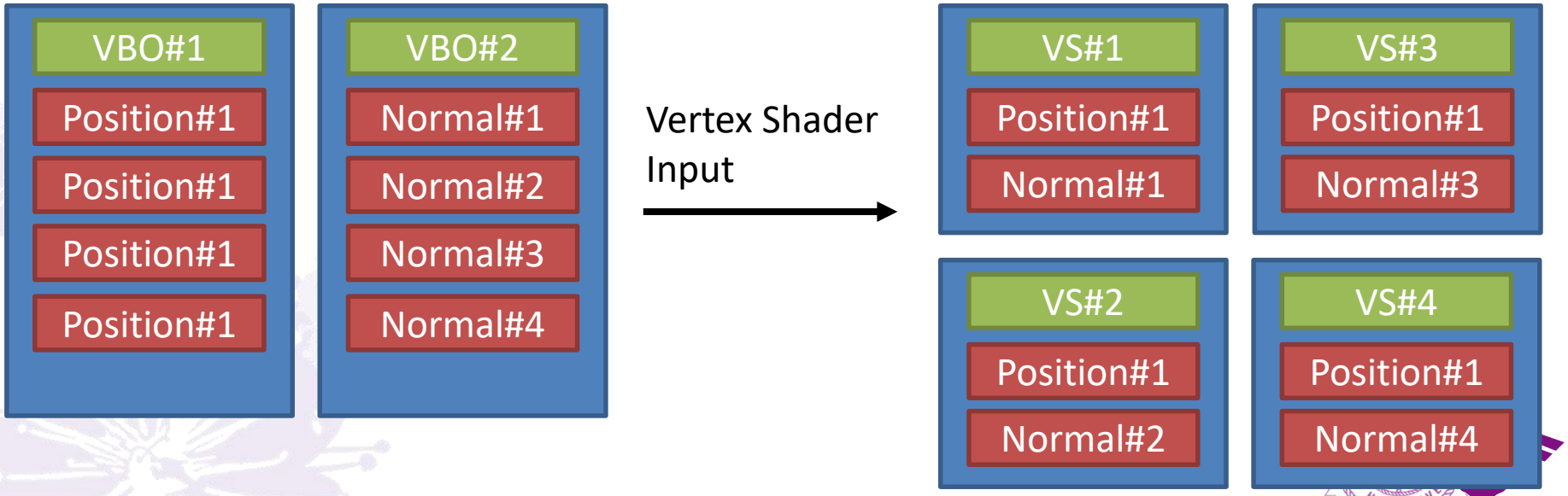
Per-face normal

- Question: can we tell OpenGL to use the same position with different normals?
- Short answer: NO

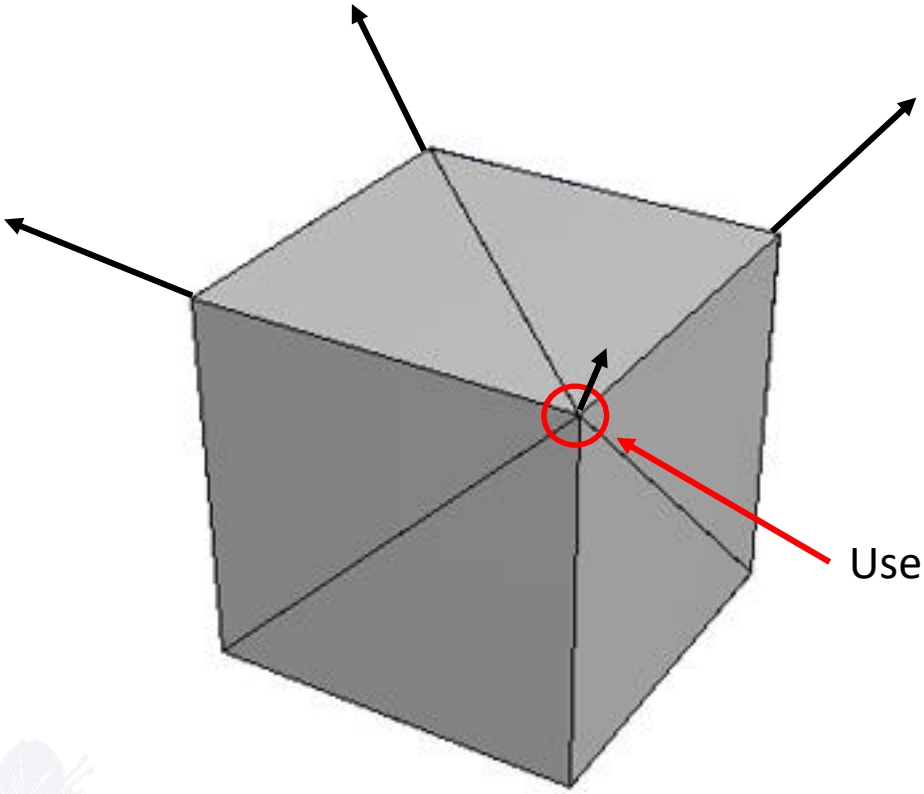


Per-face normal

- Long answer: GPUs cannot share input attributes for maximum parallel processing
- Must duplicate positions
  - No matter you or the GPU do it, it would take the same memory space anyway



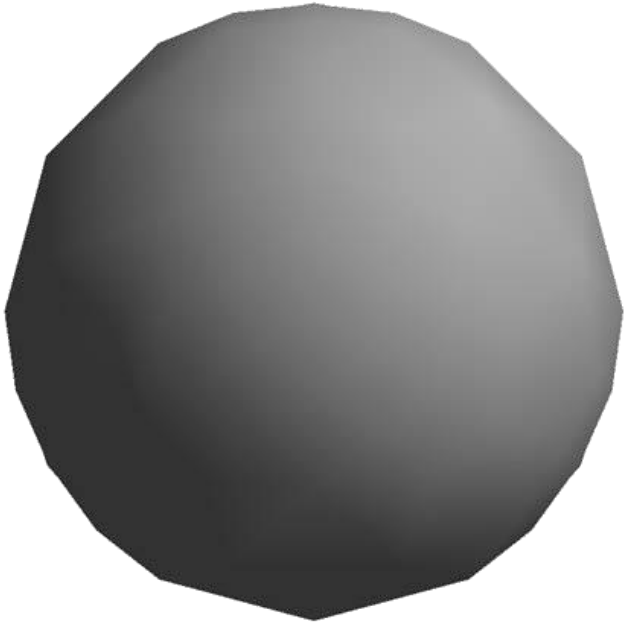
Per-vertex Normal



No problem with OpenGL, since every vertex is unique
But this can cause problems in lighting

Per-face/per-vertex normal

Per-face normal



Per-vertex normal

Per-face/per-vertex normal

- Should be clear about which you want
- Free 3D models can be given in per-face, per-vertex or even mixed, which should be treated with care

Ubisoft Grow Home uses per-face to achieve special artistic style

Assimp

- Asset Import Library
- http://assimp.sourceforge.net/
- Open source C++ 3D model format parser
- Parse & preprocess the scene

Assimp

- Describe each surface and material

```
struct Shape
{
    GLuint vao;
    GLuint vbo_position;
    GLuint vbo_normal;
    GLuint vbo_texcoord;
    GLuint ibo;
    int drawCount;
    int materialID;
};

struct Material
{
    GLuint diffuse_tex;
};
```

Assimp

- Load scene from HD

```
const aiScene *scene =
    aiImportFile(filePath, aiProcessPreset_TargetRealtime_MaxQuality);
```


Assimp

```
for (unsigned int i = 0; i < scene->mNumMaterials; ++i)
{
    aiMaterial *material = scene->mMaterials[i];
    Material material;
    aiString texturePath;
    if (material->GetTexture(aiTextureType_DIFFUSE, 0, &texturePath) ==
        aiReturn_SUCCESS)
    {
        // load width, height and data from texturePath.C_Str();
        glGenTexture(1, &material.diffuse_tex);
        glBindTexture(GL_TEXTURE_2D, material.diffuse_tex);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, width, height, 0,
                     GL_RGBA, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);
    }
    else
    {
        // load some default image as default_diffuse_tex
        material.diffuse_tex = default_diffuse_tex;
    }
    // save material…
}
```

Assimp

```
for (unsigned int i = 0; i < scene->mNumMeshes; ++i)
{
    aiMesh *mesh = scene->mMeshes[i];
    Shape shape;
    glGenVertexArrays(1, &shape.vao);
    glBindVertexArray(shape.vao);
    // create 3 vbos to hold data
    for (unsigned int v = 0; v < mesh->mNumVertices; ++v)
    {
        // mesh->mVertices[v][0~2] => position
        // mesh->mNormals[v][0~2] => normal
        // mesh->mTextureCoords[0][v][0~1] => texcoord
    }
    // create 1 ibo to hold data
    for (unsigned int f = 0; f < mesh->mNumFaces; ++f)
    {
        // mesh->mFaces[f].mIndices[0~2] => index
    }
    // glVertexAttribPointer / glEnableVertexArray calls…
    shape.materialID = mesh->mMaterialIndex;
    shape.drawCount = mesh->mNumFaces * 3;
    // save shape…
}
```

Assimp

- Release scene

```
aiReleaseImport(scene);
```


Rendering a Scene

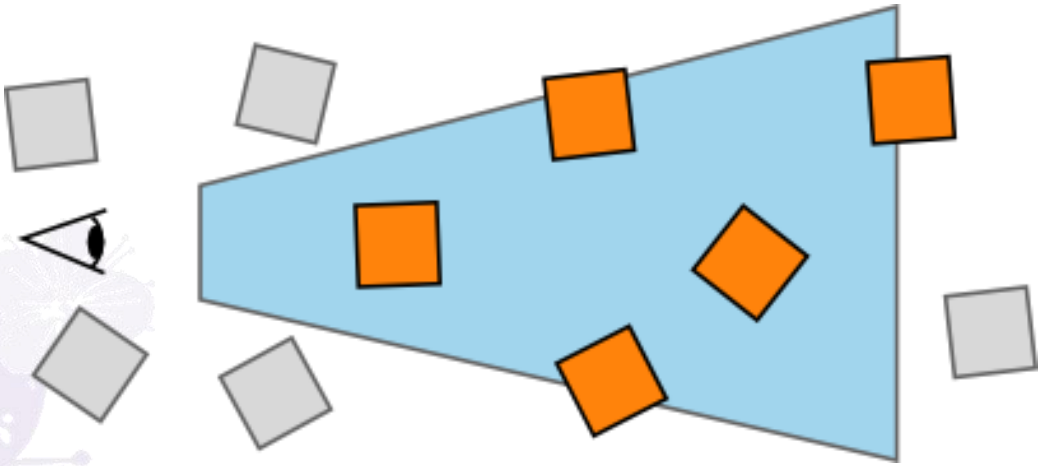
- Render the scene

```
Shape shapes[4] = { … };
Material materials[4] = { … };

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glUseProgram(program);
glUniformMatrix4fv(mv_location, 1, GL_FALSE, mv_matrix);
glUniformMatrix4fv(proj_location, 1, GL_FALSE, proj_matrix);
glActiveTexture(GL_TEXTURE0);
glUniform1i(tex_location, 0);

for(int i = 0; i < 4; ++i)
{
    glBindVertexArray(shapes[i].vao);
    int materialID = shape[i].materialID;
    glBindTexture(GL_TEXTURE_2D, materials[materialID].diffuse_tex);
    glDrawElements(GL_TRIANGLES, shapes[i].drawCount, GL_UNSIGNED_INT, 0);
}
```

ACCELERATING SCENE RENDERING

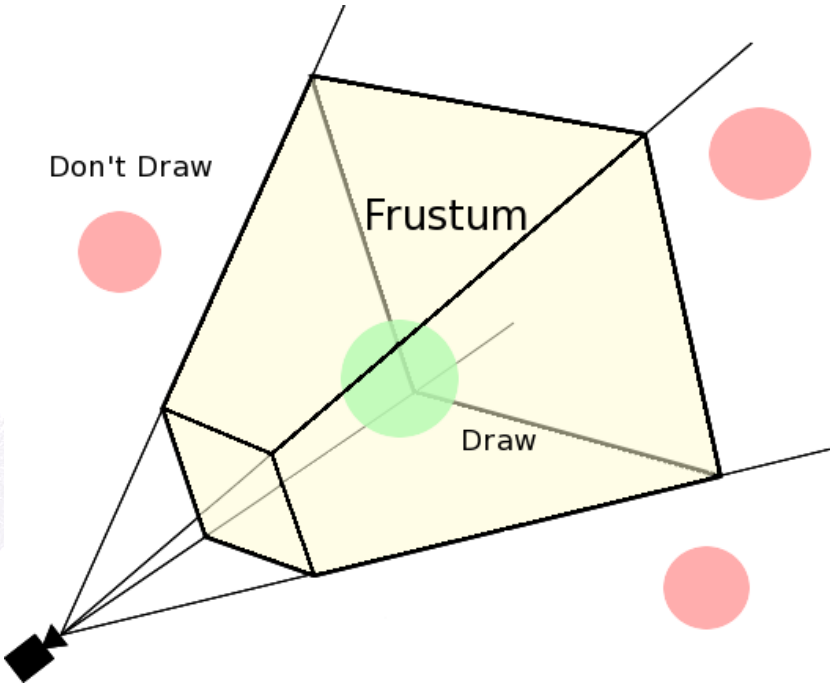
Accelerating Scene Rendering

- Recall: face culling in the rasterizer stage
- GPU & CPU time wasted
- Determine if an object is in view before drawing?



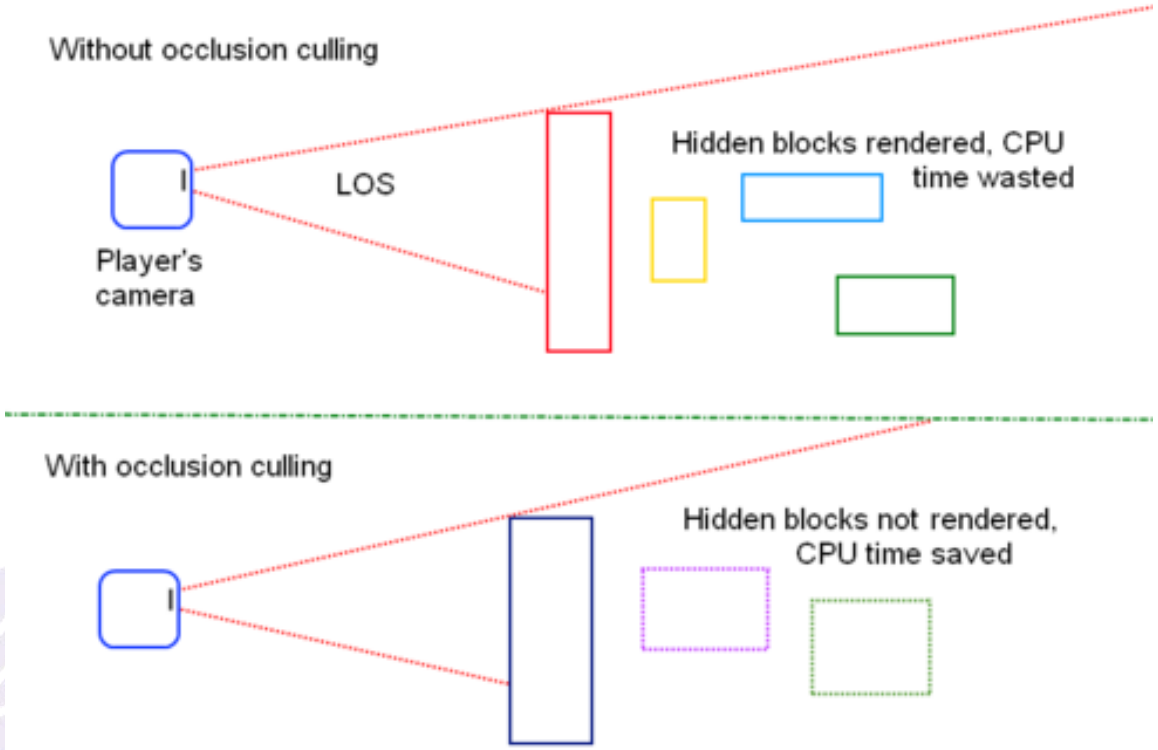
Frustum Culling

- We only see objects inside the view frustum
  - Mostly
  - Defined by view matrix and projection matrix



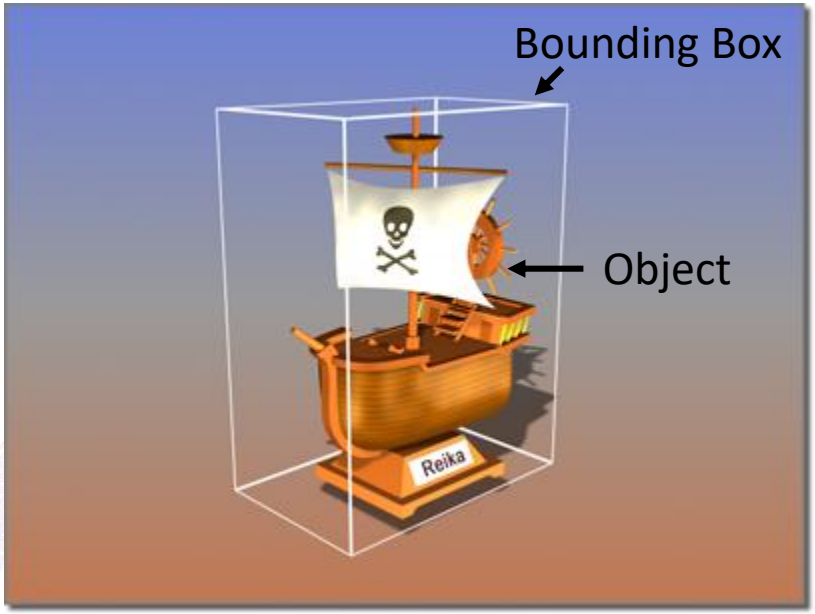
Occlusion Culling

- “Mostly”
- Occlusion culling (not covered here)



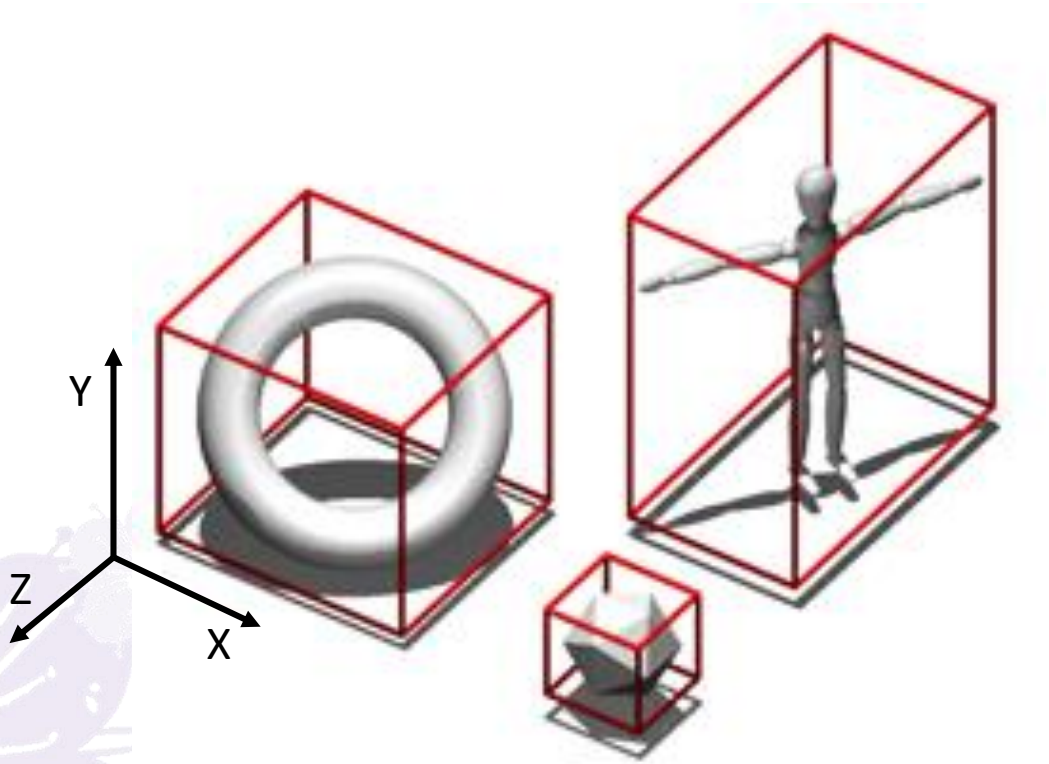
Bounding Box

- Represents a rough area of an detailed object
- Speed up computation with some “errors”

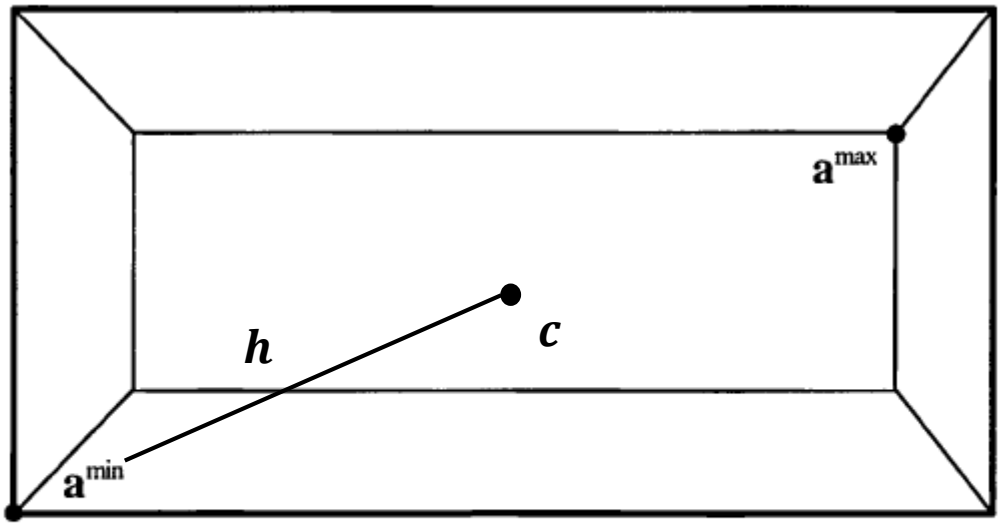


AABB

- Axis Aligned Bounding Box
  - X, Y and Z Axis



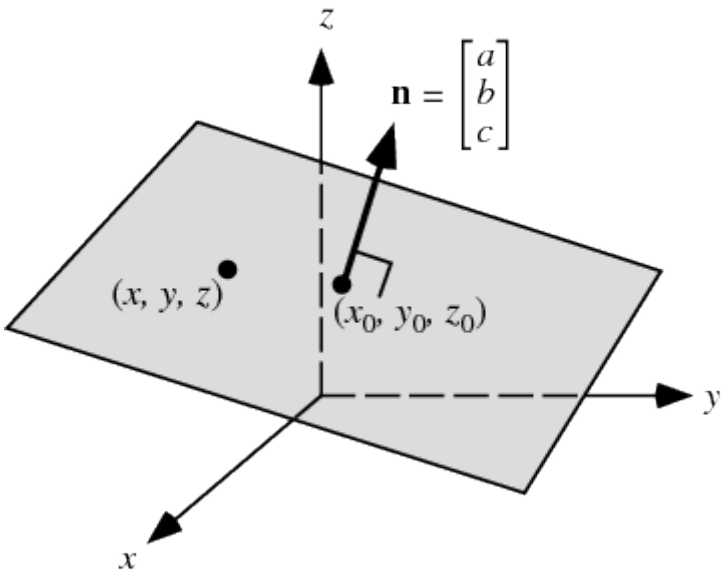
Represent an AABB



$$c = \left(a^{max} + a^{min}\right) * 0.5$$

$$h = \left(a^{max} - a^{min}\right) * 0.5$$

Plane Equation

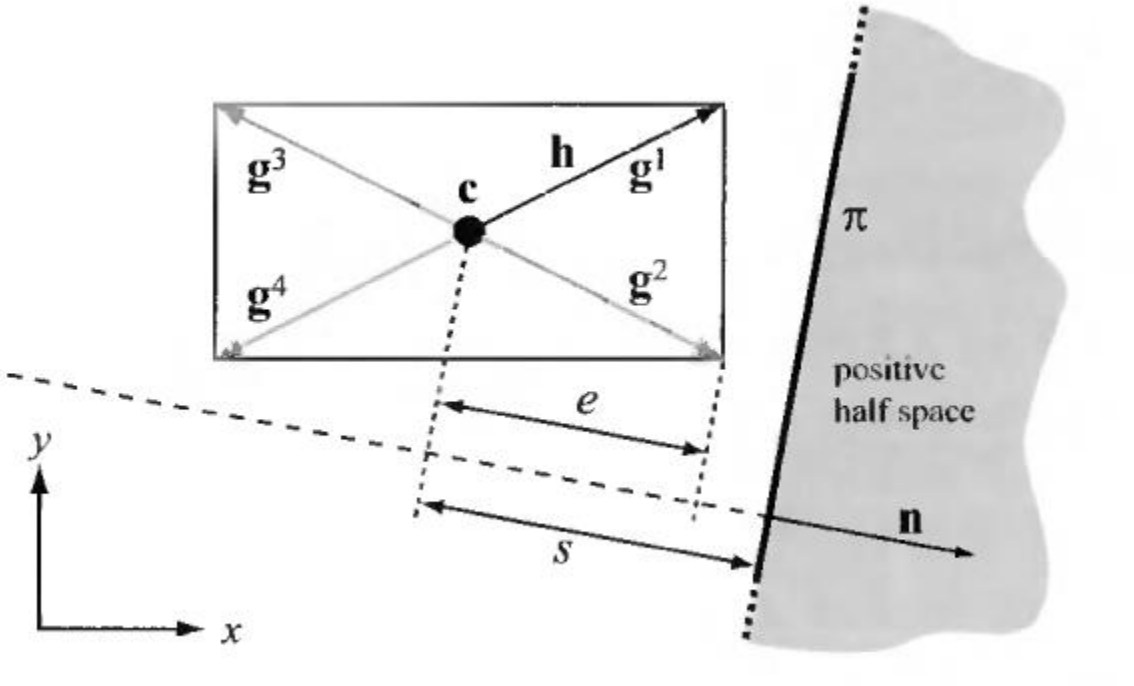


We assume that $|n| = 1$,

Plane equation is given by: $ax_0 + by_0 + cz_0 + D = 0$

The signed distance from point to plane $f(x, y, z) = ax + by + cz + D$

Plane/AABB Intersection



$$e = h_x|n_x| + h_y|n_y| + h_z|n_z|$$

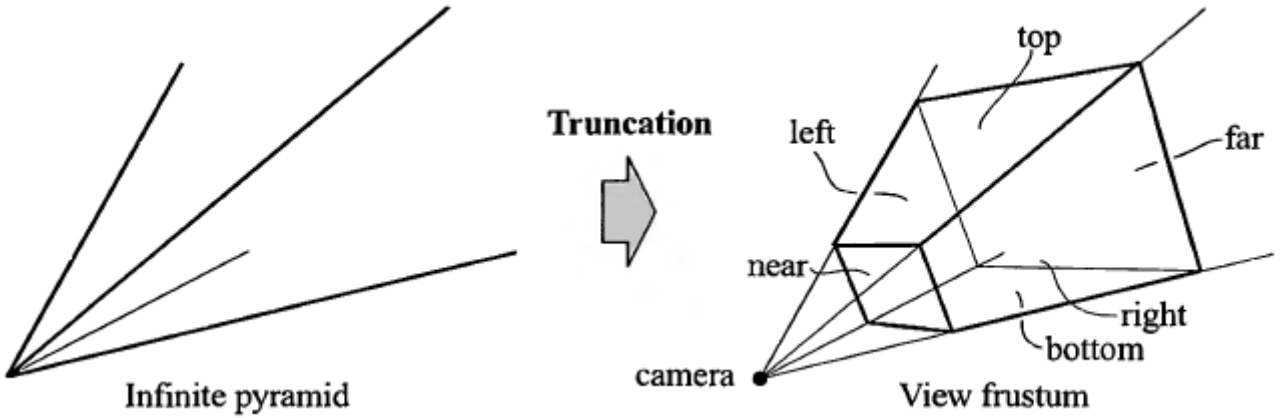
$$s = ac_x + bc_y + cc_z + D$$

e is the longest distance projected to plane normal,
if $e < -s$ or $e < s$, then we can be sure they are not intersecting

Plane/AABB Intersection

```
      bool PlaneAABBIntersect(B, π)
      returns({OUTSIDE, INSIDE, INTERSECTING});
1 :   c = (b^max + b^min)/2
2 :   h = (b^max − b^min)/2
3 :   e = h_x|n_x| + h_y|n_y| + h_z|n_z|
4 :   s = c · n + d
5 :   if(s − e > 0) return (OUTSIDE);
9 :   if(s + e < 0) return (INSIDE);
10 :  return (INTERSECTING);
```

Deriving Frustum Planes



Deriving Frustum Planes

For NDC point u, clip space point t, projection matrix P, view matrix V and world space point s:

$$u_{xyz} = \frac{t_{xyz}}{t_w} = P * V * s_{xyz}$$

If we focus on x component:

$$-1 \leq u_x \iff -1 \leq \frac{t_x}{t_w} \iff t_x + t_w \geq 0 \iff$$

$$\iff (\mathbf{m}_{0,} \cdot \mathbf{s}) + (\mathbf{m}_{3,} \cdot \mathbf{s}) \geq 0 \iff (\mathbf{m}_{0,} + \mathbf{m}_{3,}) \cdot \mathbf{s} \geq 0.$$

m_i is the i[th] row of $M = P * V$. Do this for x,y,z component and we have:

$$\begin{aligned}
-(\mathbf{m}_{3,} + \mathbf{m}_{0,}) \cdot (x, y, z, 1) = 0 \quad & [\mathbf{left}], \\
-(\mathbf{m}_{3,} - \mathbf{m}_{0,}) \cdot (x, y, z, 1) = 0 \quad & [\mathbf{right}], \\
-(\mathbf{m}_{3,} + \mathbf{m}_{1,}) \cdot (x, y, z, 1) = 0 \quad & [\mathbf{bottom}], \\
-(\mathbf{m}_{3,} - \mathbf{m}_{1,}) \cdot (x, y, z, 1) = 0 \quad & [\mathbf{top}], \\
-(\mathbf{m}_{3,} + \mathbf{m}_{2,}) \cdot (x, y, z, 1) = 0 \quad & [\mathbf{near}], \\
-(\mathbf{m}_{3,} - \mathbf{m}_{2,}) \cdot (x, y, z, 1) = 0 \quad & [\mathbf{far}].
\end{aligned}$$

Frustum/AABB Intersection

```
     bool FrustumAABBIntersect(π^0, ..., π^5, B)
     returns({OUTSIDE, INSIDE, INTERSECTING});
1 :  intersecting = false;
2 :  for k = 0 to 5
3 :     result =PlaneAABBIntersect(B, π_k)
4 :     if(result == OUTSIDE) return OUTSIDE;
5 :     elseif(result == INTERSECTING) intersecting = true;
6 :  if(intersecting == true) return INTERSECTING;
7 :  else return INSIDE;
```

Frustum Culling

- With the above mentioned, the render loop would be:

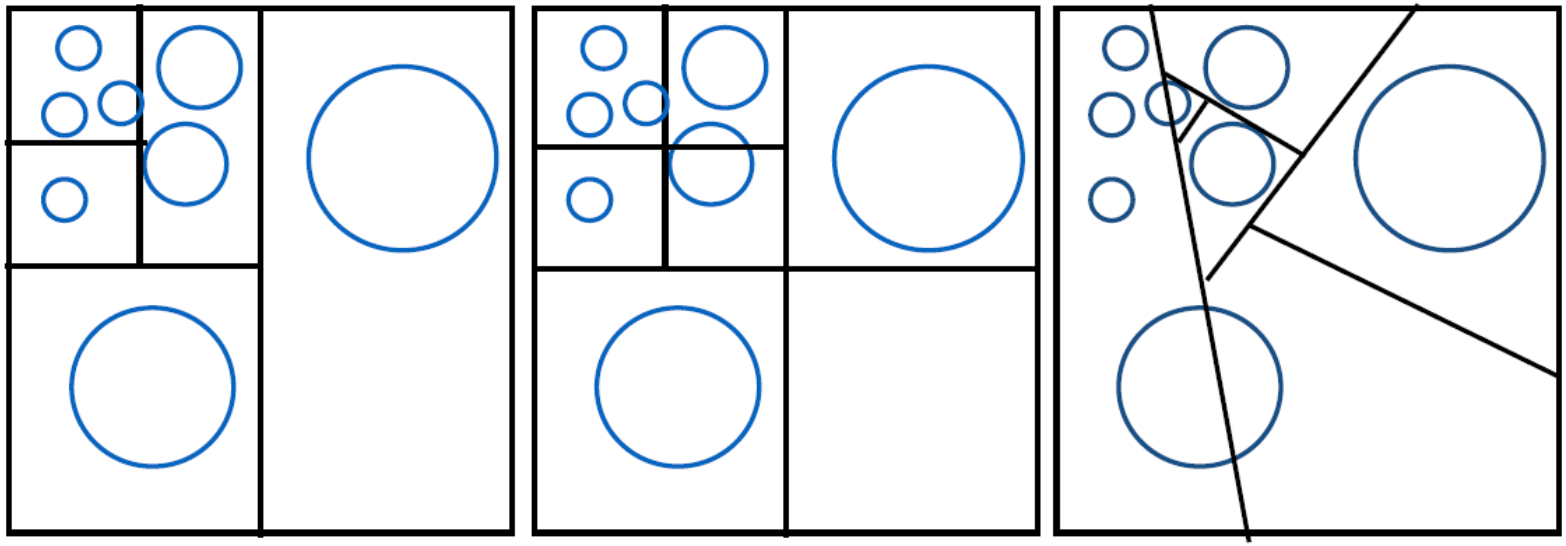
```
for(int i = 0; i < numShapes; ++i)
{
    if(FrustumAABBIntersect(shapes[i].AABB, camera.Frustum) != true)
    {
        continue;
    }
    glBindVertexArray(shapes[i].vao);
    glBindTexture(GL_TEXTURE_2D, materials[i].diffuse_tex);
    glDrawElements(GL_TRIANGLES, shapes[i].drawCount, GL_UNSIGNED_INT, 0);
}
```

Hierarchical Frustum Culling

- The above method requires $O(n)$ time to cull all objects
- Hierarchical culling: $O(\log n)$ speed up
- Any structure that shows **spatial coherence** is applicable
  - If parent is completely inside, then all children are completely inside;
  - If parent is completely outside, …

Hierarchical Frustum Culling

- Using trees, top-down
- Problem: objects that cross cells?



Hierarchical Frustum Culling

- Bounding Volume Hierarchy (BVH)
- Bottom-up