

“본 강의 동영상 및 자료는 대한민국 저작권법을 준수합니다. 본 강의 동영상 및 자료는 상명대학교 재학생들의 수업목적으로 제작·배포되는 것이므로, 수업목적으로 내려받은 강의 동영상 및 자료는 수업목적 이외에 다른 용도로 사용할 수 없으며, 다른 장소 및 타인에게 복제, 전송하여 공유할 수 없습니다. 이를 위반해서 발생하는 모든 법적 책임은 행위 주체인 본인에게 있습니다.”

# Model Loading

GPU Programming

2022학년도  
2학기



# Assimp

# Model Loading

- 단순한 컨테이너 렌더링을 벗어나는 방법?
  - 집, 차량, 사람 등... 실제 세계를 모델링한 데이터를 어떻게 그릴까?
  - 3D 아티스트가 Blender, 3DS Max, Maya와 같은 3D 모델링 툴을 사용하여 디자인한 모델을 import하면 됨 (vertex, normal, 텍스처 좌표 등..)
- 여러가지 3D 모델 포맷
  - [Wavefront .obj](#) – 간단한 material data (.mtl) 및 model data (.obj)를 저장
  - [COLLADA](#) (.dae) – Model, light, material, 애니메이션, 카메라, 완전한 scene 정보 등을 XML 형태로 저장
  - [glTF](#) – Khronos에서 만든 JSON 기반의 포맷. WebGL과 COLLADA를 결합. 다양한 데이터셋 및 압축 지원

# OBJ Example

- [bullet3/cube.obj at master · bulletphysics/bullet3 · GitHub](#)
- [bullet3/cube.mtl at master · bulletphysics/bullet3 · GitHub](#)
- s : smooth shading
- Ns: specular highlights
- Ni: index of refraction
- d: dissolve factor  
(0 – 투명, 1-불투명)
- Tr: non-transparency (= 1-d)
- Tf: transmission filter
- illum: illumination model  
(2=diffuse+specular)

```
1 # cube.obj
2 #
3
4 o cube
5 mllib cube.mtl
6
7 v -0.500000 -0.500000 0.500000
8 v 0.500000 -0.500000 0.500000
9 v -0.500000 0.500000 0.500000
10 v 0.500000 0.500000 0.500000
11 v -0.500000 0.500000 -0.500000
12 v 0.500000 0.500000 -0.500000
13 v -0.500000 -0.500000 -0.500000
14 v 0.500000 -0.500000 -0.500000
15
16 vt 0.000000 0.000000
17 vt 1.000000 0.000000
18 vt 0.000000 1.000000
19 vt 1.000000 1.000000
20
21 vn 0.000000 0.000000 1.000000
22 vn 0.000000 1.000000 0.000000
23 vn 0.000000 0.000000 -1.000000
24 vn 0.000000 -1.000000 0.000000
25 vn 1.000000 0.000000 0.000000
26 vn -1.000000 0.000000 0.000000
```

```
28 g cube
29 usemtl cube
30 s 1
31 f 1/1/1 2/2/1 3/3/1
32 f 3/3/1 2/2/1 4/4/1
33 s 2
34 f 3/1/2 4/2/2 5/3/2
35 f 5/3/2 4/2/2 6/4/2
36 s 3
37 f 5/4/3 6/3/3 7/2/3
38 f 7/2/3 6/3/3 8/1/3
39 s 4
40 f 7/1/4 8/2/4 1/3/4
41 f 1/3/4 8/2/4 2/4/4
42 s 5
43 f 2/1/5 8/2/5 4/3/5
44 f 4/3/5 8/2/5 6/4/5
45 s 6
46 f 7/1/6 1/2/6 5/3/6
47 f 5/3/6 1/2/6 3/4/6
```

```
1 newmtl cube
2 Ns 10.0000
3 Ni 1.5000
4 d 1.0000
5 Tr 0.0000
6 Tf 1.0000 1.0000 1.0000
7 illum 2
8 Ka 0.0000 0.0000 0.0000
9 Kd 0.5880 0.5880 0.5880
10 Ks 0.0000 0.0000 0.0000
11 Ke 0.0000 0.0000 0.0000
12 map_Ka cube.tga
13 map_Kd cube.png
```

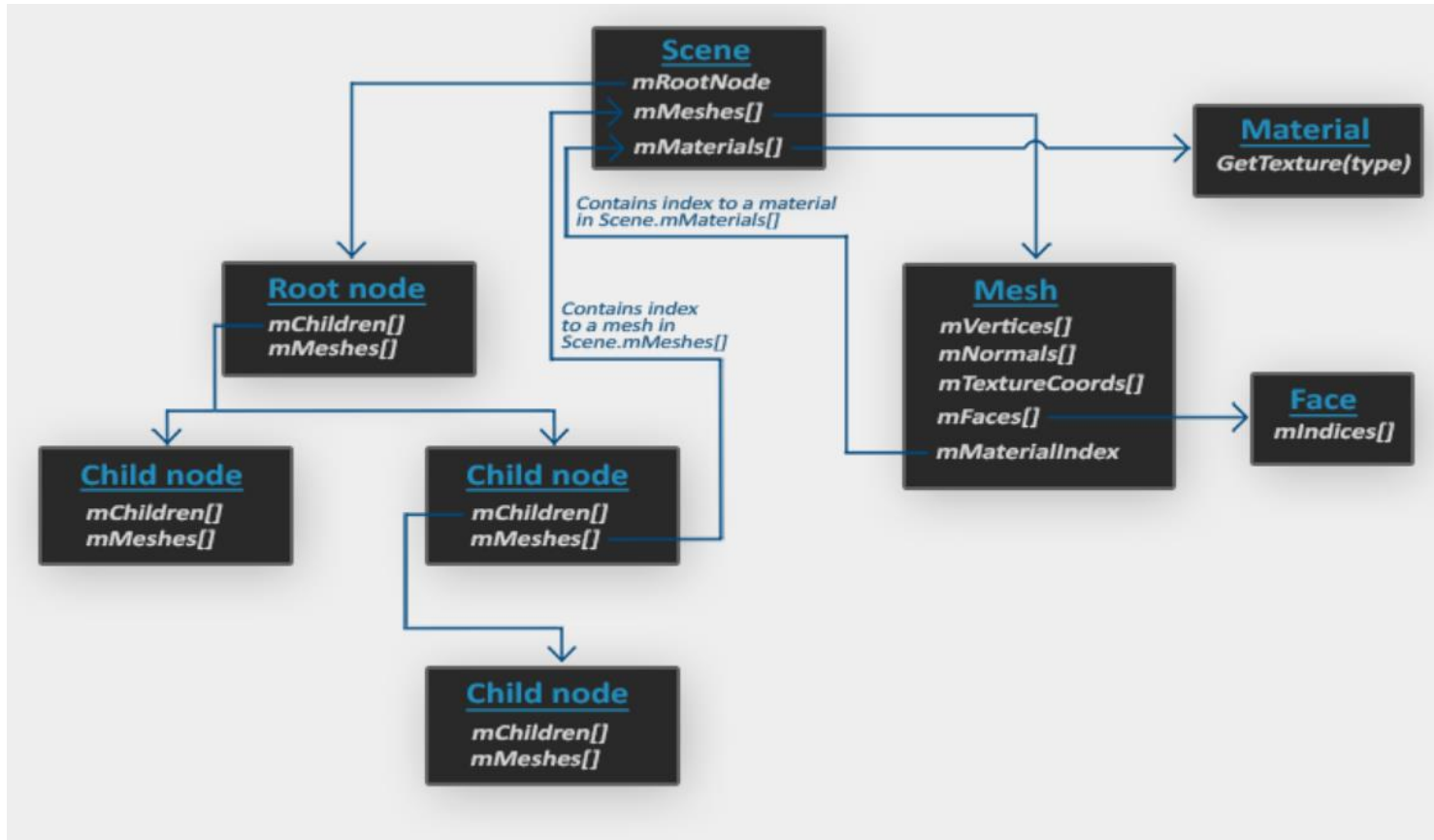


# A Model Loading Library - Assimp

- 포맷별로 loader를 만드는 수도 있지만, 잘 만들어진 라이브러리를 사용하면 편리
- [Assimp](#) (Open Asset Import Library)
  - 가장 많이 쓰이는 모델 import 라이브러리 중 하나
  - 무려 40여가지의 포맷을 지원
  - Assimp가 모델을 읽어들이면, 이를 추상화한 Assimp 데이터 구조를 통해 장면의 데이터를 얻을 수 있음

# A Model Loading Library - Assimp

- Assimp 데이터 구조의 간략한 모델



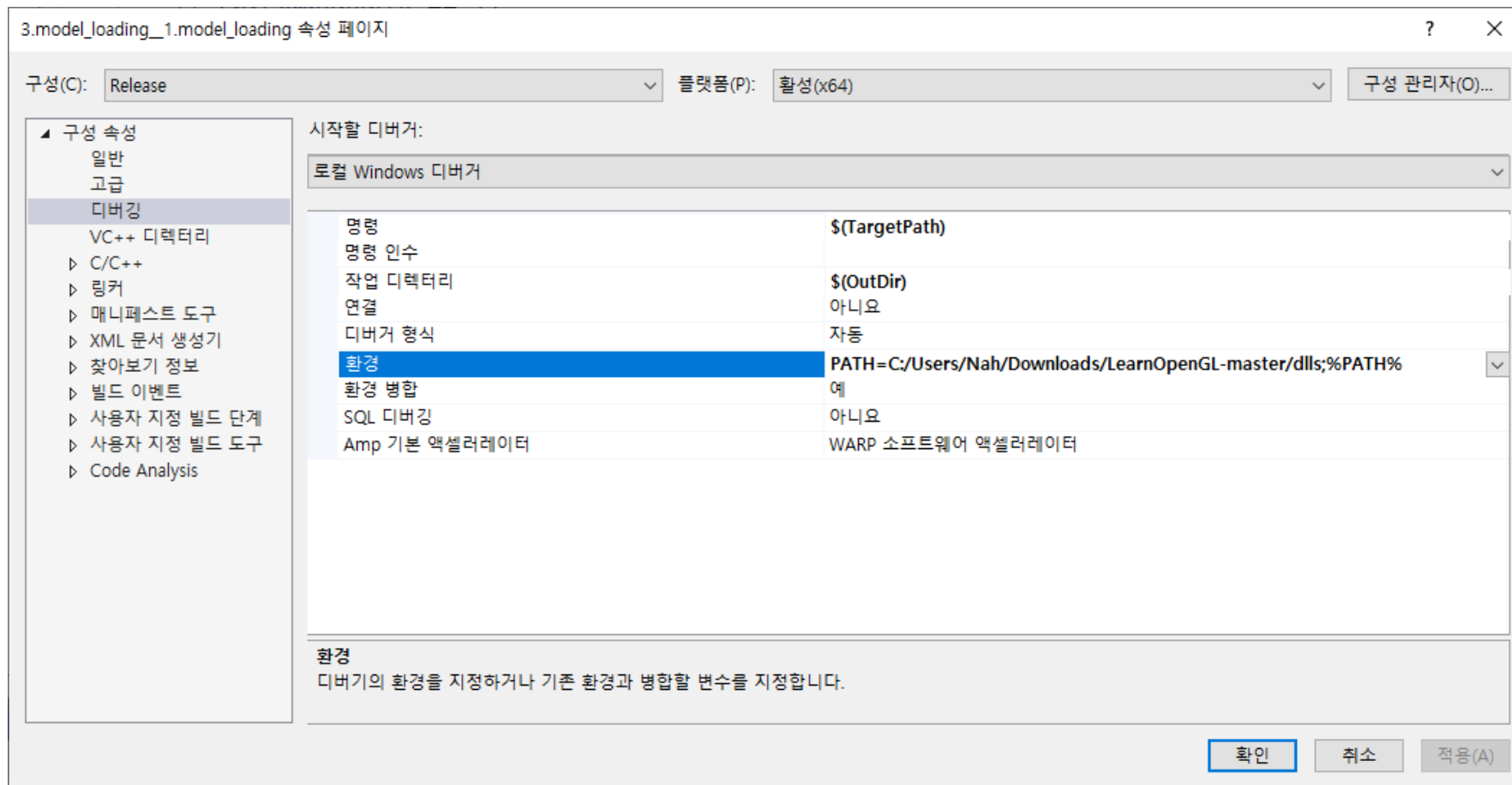
- Mesh
  - 아티스트는 보통 전체 모델을 부분부분으로 나눈 sub-model/shape으로 생성 (예: 캐릭터의 머리, 팔, 다리, 옷, 무기 등)
  - 이 각각의 shape을 mesh라 하며, 이는 OpenGL에서 객체를 그리기 위해 필요한 최소한의 것을 나타냄 (vertex 데이터, index, material 속성 등)
- 각각의 node는 변환(transform) 정보를 저장할 수도 있음
  - 계층적(hierarchical) 애니메이션 지원

# Building Assimp

- 다운로드 및 빌드 (Cmake 사용)
  - [Assimp/Build.md at master · assimp/assimp · GitHub](#)
  - 빌드 후 build/bin/Release(또는 Debug) 아래 만들어진 .dll 파일 및 build/lib/Release(또는 Debug) 아래 만들어진 .lib 파일을 본인 프로젝트로 복사 후 등록
  - 본인의 프로젝트와 똑같은 플랫폼으로 빌드 (32비트면 x86, 64비트면 x64)
- 동적 링크 라이브러리(dynamic link library)
  - 필요한 함수가 프로그램에 포함되어 컴파일되지 않고 별도 파일로 존재
  - 소스 코드에는 헤더 파일을 포함하고, lib파일은 컴파일시에 링크, dll 파일은 프로그램 구동시 불러 씀 (implicit linking 방법)
  - Dll 파일의 위치는 프로젝트 상에 PATH로 등록 (디버깅→환경→PATH=(dll이 위치한 디렉토리);%PATH%)
  - 컴파일된 실행파일을 탐색기에서 직접 실행할 때에는 dll 파일을 실행파일과 같은 디렉토리에 위치시킴. 공통으로 쓰는 dll 파일은 Windows\System32 또는 Windows\SysWOW64에 위치.

# Building Assimp

- LearnOpenGL 코드 사용시
  - Cmake에서 절대경로로 dll이 위치한 디렉토리를 프로젝트 상에 잡아 줌







# Mesh

# Mesh Class

- Assimp의 데이터 구조를 OpenGL이 이해할 수 있는 포맷으로 변환할 필요가 있음
- Mesh 클래스
  - 위치 벡터, 노멀 벡터, 텍스처 좌표 등을 포함하는 vertex들의 모음
  - 인덱싱을 위한 index, 텍스처 형태의 material 데이터도 포함 가능

```
struct Vertex {  
    glm::vec3 Position;  
    glm::vec3 Normal;  
    glm::vec2 TexCoords;  
};
```

```
struct Texture {  
    unsigned int id;  
    string type;  
};
```

```
class Mesh {  
public:  
    // mesh data  
    vector<Vertex>      vertices;  
    vector<unsigned int> indices;  
    vector<Texture>     textures;  
  
    Mesh(vector<Vertex> vertices, vector<unsigned int> indices, vector<Texture> textures);  
    void Draw(Shader &shader);  
private:  
    // render data  
    unsigned int VAO, VBO, EBO;  
  
    void setupMesh();  
};
```

# Initialization

- 생성자

```
Mesh(vector<Vertex> vertices, vector<unsigned int> indices, vector<Texture> textures)
{
    this->vertices = vertices;
    this->indices = indices;
    this->textures = textures;

    setupMesh();
}
```

- SetupMesh() 함수

- 적절한 버퍼 설정
- Vertex attribute pointer를 통해 vertex shader의 layout 지정

```
void setupMesh()
{
    glGenVertexArrays(1, &VA0);
    glGenBuffers(1, &VB0);
    glGenBuffers(1, &EB0);

    glBindVertexArray(VA0);
    glBindBuffer(GL_ARRAY_BUFFER, VB0);

    glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), &vertices[0], GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EB0);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.size() * sizeof(unsigned int),
                 &indices[0], GL_STATIC_DRAW);

    // vertex positions
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)0);
    // vertex normals
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Normal));
    // vertex texture coords
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, TexCoords));

    glBindVertexArray(0);
}
```

# Initialization

- C/C++ 구조체의 속성들은 순차적으로 메모리에 저장

```
Vertex vertex;  
vertex.Position = glm::vec3(0.2f, 0.4f, 0.6f);  
vertex.Normal    = glm::vec3(0.0f, 1.0f, 0.0f);  
vertex.TexCoords = glm::vec2(1.0f, 0.0f);  
// = [0.2f, 0.4f, 0.6f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f];
```

```
struct Vertex {  
    glm::vec3 Position;  
    glm::vec3 Normal;  
    glm::vec2 TexCoords;  
};  
  
class Mesh {  
public:  
    // mesh data  
    vector<Vertex> vertices;  
    vector<unsigned int> indices;  
    vector<Texture> textures;
```

- 그 덕에, Vertex 구조체 벡터의 포인터(아래에서는 vertices[0])를 glBufferData에 바로 전달 가능

```
glBufferData(GL_ARRAY_BUFFER, vertices.size() * sizeof(Vertex), vertices[0], GL_STATIC_DRAW);
```

- stddef.h에 정의된 offsetof(s,m)이라는 매크로 함수 이용
  - 구조체 s에서 변수 m이 메모리상에서 얼마나 떨어져 있는지 알 수 있음
  - 이를 이용해 glVertexAttribPointer() 함수의 offset 파라미터를 정의 가능
  - 아래 예에서는 offset이 12바이트로 설정됨 (3 floats \* 4 bytes)

```
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, sizeof(Vertex), (void*)offsetof(Vertex, Normal));
```

- 또 다른 vertex attribute를 원하더라도, 간단히 struct에 추가 가능

# Rendering

- Draw 함수 설계

- glDrawElements() 함수를 호출하여 mesh를 실제로 렌더링하기 전 적절한 텍스처를 바인딩해야 함
- 이를 수월하게 수행하기 위해 특별한 네이밍 관습 적용 – texture\_typeN (type: diffuse, specular 등)
- 3개 diffuse, 2개 specular 텍스처를 5개의 텍스처 유닛에 바인딩하고, 각 유닛의 번호를 shader uniform에 연결시켜주는 예제

FS

```
uniform sampler2D texture_diffuse1;  
uniform sampler2D texture_diffuse2;  
uniform sampler2D texture_diffuse3;  
uniform sampler2D texture_specular1;  
uniform sampler2D texture_specular2;
```

CPU

```
void Draw(Shader &shader)  
{  
    unsigned int diffuseNr = 1;  
    unsigned int specularNr = 1;  
    for(unsigned int i = 0; i < textures.size(); i++)  
    {  
        glActiveTexture(GL_TEXTURE0 + i); // activate proper texture unit before binding  
        // retrieve texture number (the N in diffuse_textureN)  
        string number;  
        string name = textures[i].type;  
        if(name == "texture_diffuse")  
            number = std::to_string(diffuseNr++);  
        else if(name == "texture_specular")  
            number = std::to_string(specularNr++);  
  
        shader.setInt(("material." + name + number).c_str(), i);  
        glBindTexture(GL_TEXTURE_2D, textures[i].id);  
    }  
    glActiveTexture(GL_TEXTURE0);  
  
    // draw mesh  
    glBindVertexArray(VAO);  
    glDrawElements(GL_TRIANGLES, indices.size(), GL_UNSIGNED_INT, 0);  
    glBindVertexArray(0);  
}
```



# Model



# Model Class

- 여러 mesh들을 가지고 있는 전체적인 모델에 대한 클래스
  - 예) 나무로 된 발코니, 타워, 수영장을 가지고 있는 집
  - Assimp를 통해 model을 읽어들인 후 Mesh 객체로 변환
- Model class

```
class Model
{
    public:
        Model(char *path)
        {
            loadModel(path);
        }
        void Draw(Shader &shader);
    private:
        // model data
        vector<Mesh> meshes;
        string directory;

        void loadModel(string path);
        void processNode(aiNode *node, const aiScene *scene);
        Mesh processMesh(aiMesh *mesh, const aiScene *scene);
        vector<Texture> loadMaterialTextures(aiMaterial *mat, aiTextureType type,
                                              string typeName);
};

void Draw(Shader &shader)
{
    for(unsigned int i = 0; i < meshes.size(); i++)
        meshes[i].Draw(shader);
}
```

# Importing a 3D Model into OpenGL

- Assimp 관련 헤더를 먼저 include

```
#include <assimp/Importer.hpp>
#include <assimp/scene.h>
#include <assimp/postprocess.h>
```

- loadModel() 함수

```
void loadModel(string path)
{
    Assimp::Importer import;
    const aiScene *scene = import.ReadFile(path, aiProcess_Triangulate | aiProcess_FlipUVs);

    if(!scene || scene->mFlags & AI_SCENE_FLAGS_INCOMPLETE || !scene->mRootNode)
    {
        cout << "ERROR::ASSIMP::" << import.GetErrorString() << endl;
        return;
    }
    directory = path.substr(0, path.find_last_of('/'));

    processNode(scene->mRootNode, scene);
}
```

- ReadFile에 그 외 사용 가능한 post-processing 옵션에는 aiProcess\_GenNormals, aiProcess\_SplitLargeMeshes, aiProcess\_OptimizeMeshes 등이 있음

[Assimp: postprocess.h File Reference \(sourceforge.net\)](http://sourceforge.net/projects/assimp/files/doc/postprocess.h)

# Importing a 3D Model into OpenGL

- processNode() 함수

```
void processNode(aiNode *node, const aiScene *scene)
{
    // process all the node's meshes (if any)
    for(unsigned int i = 0; i < node->mNumMeshes; i++)
    {
        aiMesh *mesh = scene->mMeshes[node->mMeshes[i]];
        meshes.push_back(processMesh(mesh, scene));
    }
    // then do the same for each of its children
    for(unsigned int i = 0; i < node->mNumChildren; i++)
    {
        processNode(node->mChildren[i], scene);
    }
}
```

# Assimp to Mesh

- processMesh() 함수

```
Mesh processMesh(aiMesh *mesh, const aiScene *scene)
```

```
{  
    vector<Vertex> vertices;  
    vector<unsigned int> indices;  
    vector<Texture> textures;  
  
    for(unsigned int i = 0; i < mesh->mNumVertices; i++)  
    {  
        Vertex vertex;  
        // process vertex positions, normals and texture coordinates  
        [...]  
        vertices.push_back(vertex);  
    }  
    // process indices  
    [...]  
    // process material  
    if(mesh->mMaterialIndex >= 0)  
    {  
        [...]  
    }  
  
    return Mesh(vertices, indices, textures);  
}
```

```
glm::vec3 vector;  
vector.x = mesh->mVertices[i].x;  
vector.y = mesh->mVertices[i].y;  
vector.z = mesh->mVertices[i].z;  
vertex.Position = vector;  
vector.x = mesh->mNormals[i].x;  
vector.y = mesh->mNormals[i].y;  
vector.z = mesh->mNormals[i].z;  
vertex.Normal = vector;
```

```
if(mesh->mTextureCoords[0]) // does the mesh contain texture coordinates?  
{  
    glm::vec2 vec;  
    vec.x = mesh->mTextureCoords[0][i].x;  
    vec.y = mesh->mTextureCoords[0][i].y;  
    vertex.TexCoords = vec;  
}  
else  
    vertex.TexCoords = glm::vec2(0.0f, 0.0f);
```

# Assimp to Mesh

- processMesh() 함수

```
Mesh processMesh(aiMesh *mesh, const aiScene *scene)
{
    vector<Vertex> vertices;
    vector<unsigned int> indices;
    vector<Texture> textures;

    for(unsigned int i = 0; i < mesh->mNumVertices; i++)
    {
        Vertex vertex;
        // process vertex positions, normals and texture coordinates
        [...]
        vertices.push_back(vertex);
    }
    // process indices
    [...]
    // process material
    if(mesh->mMaterialIndex >= 0)
    {
        [...]
    }

    return Mesh(vertices, indices, textures);
}
```

```
for(unsigned int i = 0; i < mesh->mNumFaces; i++)
{
    aiFace face = mesh->mFaces[i];
    for(unsigned int j = 0; j < face.mNumIndices; j++)
        indices.push_back(face.mIndices[j]);
}
```

- 여기서는 aiProcess\_Triangulate 덕에,  
각 face 별로 하나의  
primitive(triangle)를 가짐

# Assimp to Mesh

- processMesh() 함수

```
Mesh processMesh(aiMesh *mesh, const aiScene *scene)
{
    vector<Vertex> vertices;
    vector<unsigned int> indices;
    vector<Texture> textures;

    for(unsigned int i = 0; i < mesh->mNumVertices; i++)
    {
        Vertex vertex;
        // process vertex positions, normals and texture coordinates
        [...]
        vertices.push_back(vertex);
    }
    // process indices
    [...]
    // process material
    if(mesh->mMaterialIndex >= 0)
    {
        [...]
    }
}

if(mesh->mMaterialIndex >= 0)
{
    aiMaterial *material = scene->mMaterials[mesh->mMaterialIndex];
    vector<Texture> diffuseMaps = loadMaterialTextures(material,
                                                         aiTextureType_DIFFUSE, "texture_diffuse");
    textures.insert(textures.end(), diffuseMaps.begin(), diffuseMaps.end());
    vector<Texture> specularMaps = loadMaterialTextures(material,
                                                         aiTextureType_SPECULAR, "texture_specular");
    textures.insert(textures.end(), specularMaps.begin(), specularMaps.end());
}

return Mesh(vertices, indices, textures);
}
```



# Assimp to Mesh

- loadMaterialTextures() 함수
  - GetTexture() 함수를 통해 텍스처 파일 경로를 얻어옴
  - textureFromFile() 함수를 통해 텍스처를 직접 읽어들이고 textures 벡터에 등록
  - 모델과 텍스처는 동일한 위치에 있고, 텍스처 경로는 상대경로로 등록되어 있다고 가정

```
vector<Texture> loadMaterialTextures(aiMaterial *mat, aiTextureType type, string typeName)
{
    vector<Texture> textures;
    for(unsigned int i = 0; i < mat->GetTextureCount(type); i++)
    {
        aiString str;
        mat->GetTexture(type, i, &str);
        Texture texture;
        texture.id = TextureFromFile(str.C_Str(), directory);
        texture.type = typeName;
        texture.path = str;
        textures.push_back(texture);
    }
    return textures;
}
```

# An Optimization

- loadMaterialTextures() 함수 수정
  - 하나의 텍스처가 여러 material에 포함되어 있을 경우, 이미 읽어 들인 텍스처를 또 읽어 들일 필요는 없음
  - 이미 읽어 들인 텍스처를 저장한 texture\_loaded에 존재하지 않는 텍스처만 새로 읽어 들이도록 변경

```
struct Texture {  
    unsigned int id;  
    string type;  
    string path; // we store the path of the texture  
};
```

```
vector<Texture> textures_loaded;
```

```
vector<Texture> loadMaterialTextures(aiMaterial *mat, aiTextureType type, string typeName)  
{  
    vector<Texture> textures;  
    for(unsigned int i = 0; i < mat->GetTextureCount(type); i++)  
    {  
        aiString str;  
        mat->GetTexture(type, i, &str);  
        bool skip = false;  
        for(unsigned int j = 0; j < textures_loaded.size(); j++)  
        {  
            if(std::strcmp(textures_loaded[j].path.data(), str.C_Str()) == 0)  
            {  
                textures.push_back(textures_loaded[j]);  
                skip = true;  
                break;  
            }  
        }  
        if(!skip)  
        {  
            // if texture hasn't been loaded already, load it  
            Texture texture;  
            texture.id = TextureFromFile(str.C_Str(), directory);  
            texture.type = typeName;  
            texture.path = str.C_Str();  
            textures.push_back(texture);  
            textures_loaded.push_back(texture); // add to loaded textures  
        }  
    }  
    return textures;  
}
```

# No More Containers!

- [Survival Guitar Backpack - Download Free 3D model by Berk Gedik \(@berkgedik\) \[799f8c4\] \(sketchfab.com\)](https://sketchfab.com/berkgedik/3d-models/survival-guitar-backpack) 모델을 읽어 렌더링한 예
  - backpack.obj를 읽는 Model 객체를 선언
  - diffuse texture map을 입혀 보여주는 shader를 적용 (왼쪽 및 중간. 중간은 wireframe 모드)
  - 추가로 specular texture map과 point light를 통한 lighting을 적용 (오른쪽)

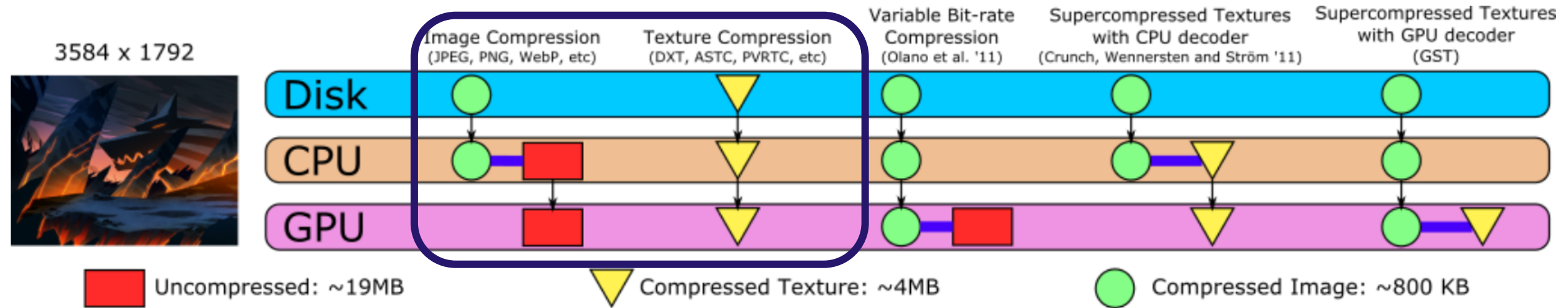




# Texture Compression

# 텍스처 압축의 필요성

- 지금까지는 텍스처를 올릴 때 .jpg 형식으로 된 파일을 많이 이용했지만, GPU에 올릴 때에는 stb\_image.h와 같은 라이브러리에서 이를 풀어서 올리게 됨

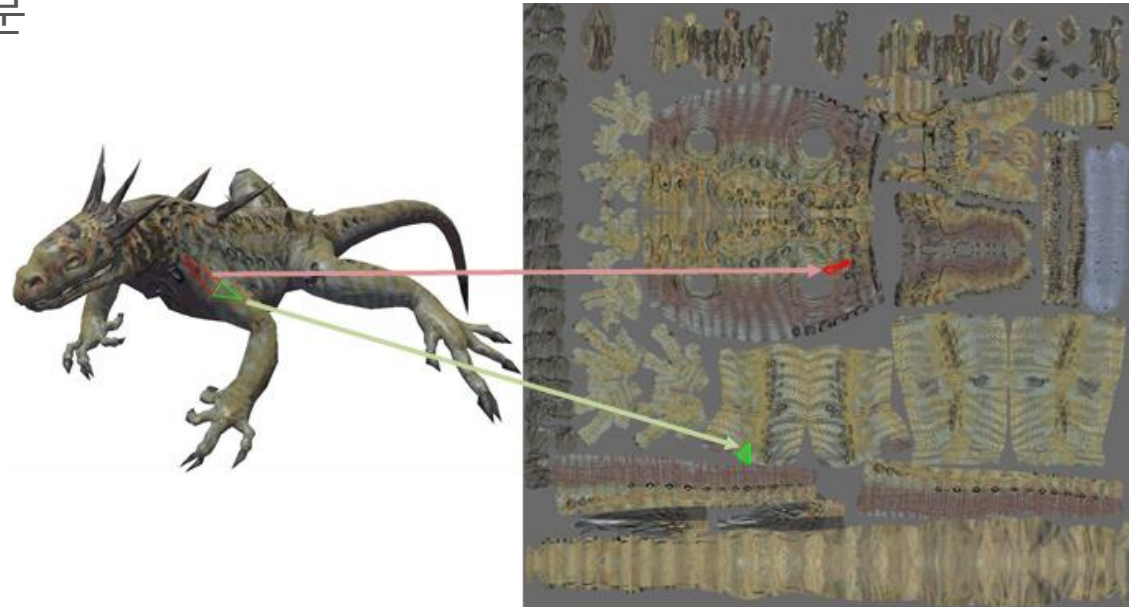


[GST: GPU-decodable Supercompressed Textures \(unc.edu\)](http://unc.edu)

- 이에 따라 성능 저하 및 GPU 메모리 사용량 증가와 같은 문제점 발생
- 따라서 대부분의 경우 GPU가 디코딩 가능한 압축된 형태로 텍스처를 올리게 됨

# 텍스처 압축의 특성

- 디코딩은 GPU 상에서 실시간에 처리되어야 하지만, 인코딩은 보통 CPU에서 미리 수행
  - GPU 안에 디코더가 내장되어 있으며, 이 내장된 디코더의 종류는 GPU마다 다름
- 랜덤 액세스가 가능해야 함
  - 텍스처 좌표에 따라 어느 텍셀을 가져올지 모르기 때문
  - 기존 JPEG과 같은 이미지 압축과 가장 다른 점
- 손실 압축 방식 사용
  - 비손실 압축보다는 압축률이 높고 손실 발생
  - 기존 손실 이미지 압축(JPEG 등)보다는 동일 품질에서 압축률이 낮고, 다소 인코딩에 시간이 오래 걸림



[Texture Compression Techniques \(sv-journal.org\)](http://sv-journal.org)

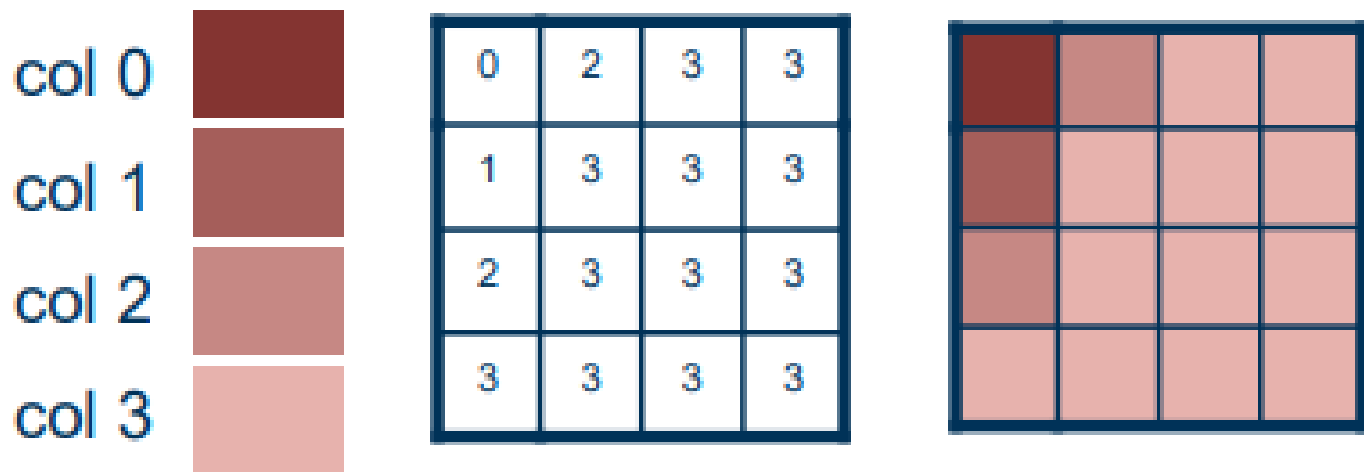


# 대표적인 텍스처 압축 형식

- BC (Block Compression, aka DXT)
  - 마이크로소프트에서 관리하는 DirectX 표준 포맷. OpenGL에서도 사용 가능
  - 윈도우 기반 데스크탑 앱은 대부분 이 포맷 사용
  - 압축 품질 및 지원 범위(알파 채널, HDR 등)에 따라 BC1, BC2, BC3, BC4, BC5, BC6H, BC7로 나뉨
- PVRTC (PowerVR Texture Compression)
  - 과거 Apple AP에 GPU IP를 납품하던 PowerVR사에서 만든 포맷
  - 과거 iOS 표준
- ETC (Ericsson Texture Compression)
  - 에릭슨사에서 만들어 Khronos에 기증한 포맷
  - 압축 품질 및 지원 범위에 따라 ETC1, ETC2, EAC로 나뉨
  - 안드로이드의 사실상 표준 (OpenGL ES 2.0 및 3.0 표준)
- ASTC (Adaptive Scalable Texture Compression)
  - ARM사에서 만들어 Khronos에 기증한 포맷
  - 다양한 텍스처 형태를 지원하고, 압축 품질/압축률을 조정 가능
  - 최신 iOS, 안드로이드 기기에서 지원 (OpenGL ES 3.2 이상)

# 텍스처 압축의 원리

- BC1 (aka S3TC, DXT1)
  - 4x4 블록별로, endpoint 컬러 2개를 구함 (col 0, col 3)
  - 이 두 컬러를 보간한 컬러 2개를 더 구함 (col 1, col 2)
  - 각 픽셀별로, 이 4개 color 중 어떤 것을 선택할지 결정
  - 두 endpoint 컬러에 32비트(2x16비트), 인덱스 테이블에 32비트(16픽셀x2비트) 할당
  - 압축률은 6:1 – 압축 전 24비트 컬러x16픽셀 = 384비트 → 압축 후 64비트

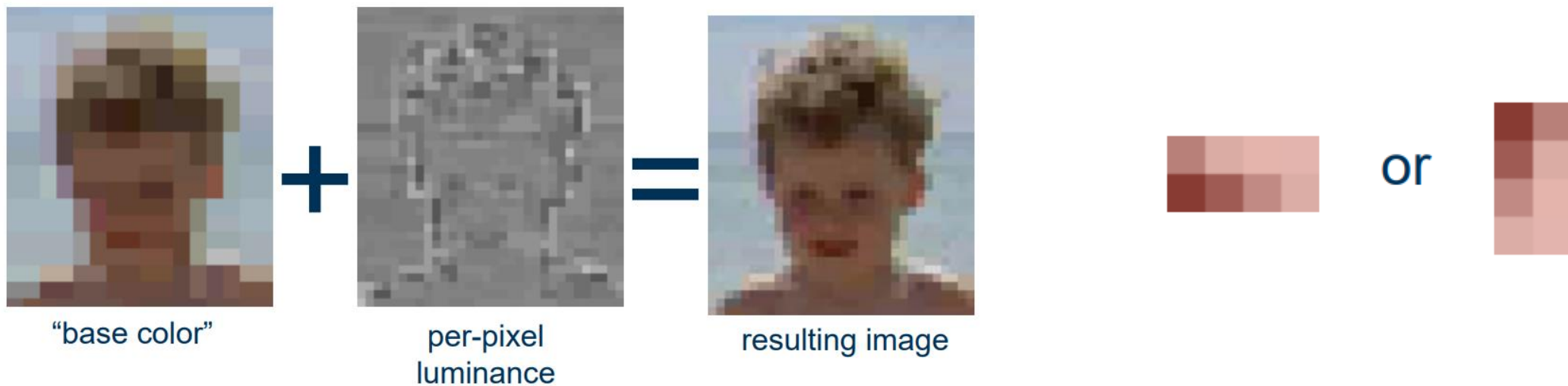


[ETC2: Texture Compression Using Invalid Combinations \(graphicshardware.org\)](http://graphicshardware.org)

# 텍스처 압축의 원리

- ETC1

- DXT1와 블록 크기(4x4) 및 압축률(6:1) 동일
- 4x2 또는 2x4 서브블록별로 base color를 구한 후, 픽셀별로 luminance 변조값을 구함
- 디코딩시에는 이 base color와 per-pixel luminance를 합쳐 최종 픽셀 색상을 구함
- 총 4개의 조합 존재 – flip (4x2 혹은 2x4?) 및 diff (두 base color가 의존적 혹은 독립적?) flag에 따름



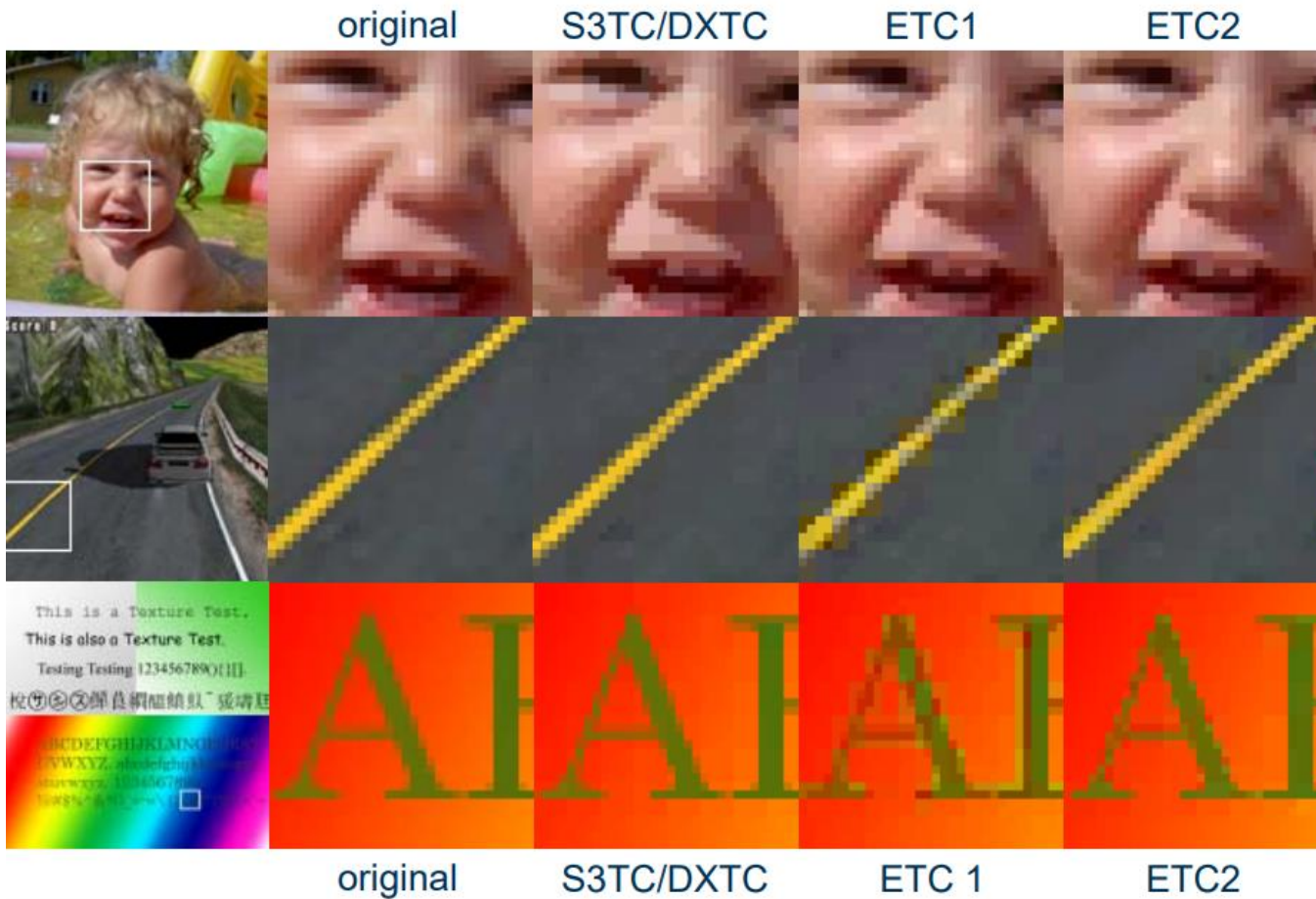
- ETC2

- ETC1에 계단 현상을 완화하는 T/H모드와 부드러움을 더해주는 planar 모드를 추가하고, 알파값 압축 지원

[ETC2: Texture Compression Using Invalid Combinations \(graphicshardware.org\)](http://graphicshardware.org)

# 텍스처 압축의 원리

- 압축 품질 비교















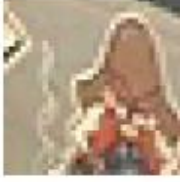

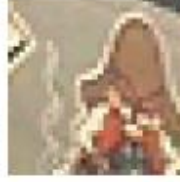












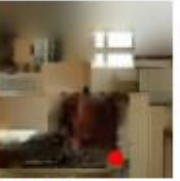






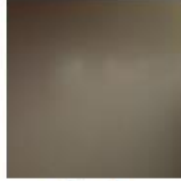

[ETC2: Texture Compression Using Invalid Combinations \(graphicshardware.org\)](http://graphicshardware.org)

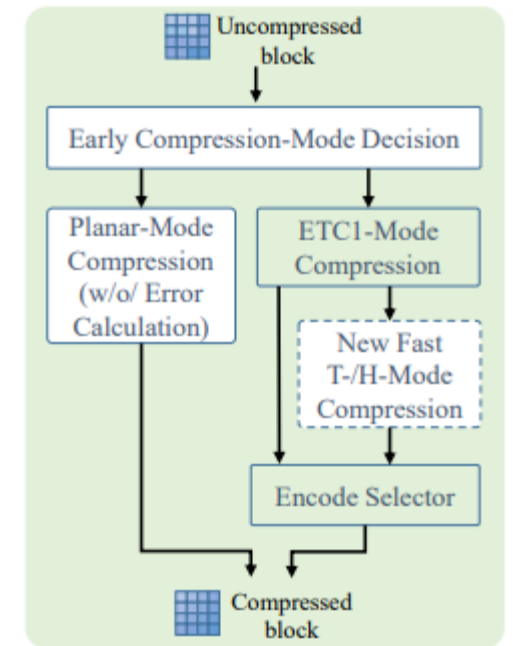
# 압축된 텍스처의 사용 방법

- 사용하고자 하는 압축 코덱을 지원하는 인코더를 이용, 이미지를 압축
  - [GitHub – wolfpld/etcpak: The fastest ETC compressor on the planet](#)
  - [Compressorator – GPUOpen](#)
- .dds (DirectDrawSurface) 또는 .ktx (Khronos Texture) 형태로 저장된 텍스처를 전용 로더로 읽어들이м
  - [GitHub – paroj/nv\\_dds: DDS image loader for OpenGL/ OpenGL ES2](#)
  - [GitHub – DeanoC/tiny\\_ktx: Small C based KTX texture loader \(inspired by syoyo tiny libraries\)](#)
- glTexImage2D() 대신 glCompressedTexImage2D()와 같은 함수를 이용하여 텍스처를 GPU 메모리에 적재
  - internalformat 파라미터에 텍스처 포맷 지정  
void glCompressedTexImage2D(GLenum target, GLint level, GLenum internalformat, GLsizei width, GLsizei height, GLint border, GLsizei imageSize, const void \* data);
  - [glCompressedTexImage2D – OpenGL ES 3.2 Reference Pages \(khronos.org\)](#)
- glGenerateMipmap() 함수를 이용한 실시간 mip맵 생성은 불가



# 고속 텍스처 압축 알고리즘 - QuickETC2

	Original	etcpak ETC1	etcpak ETC2 (p)	Ours ETC2 (p)	Ours ETC2	Etc2comp ETC2	ETCPACK ETC2	astcenc ASTC 6x6
 Kodim05 (768×512)		 0.14 ms	 0.20 ms	 0.14 ms	 0.27 ms	 105 ms	 380 ms	 43 ms
 Kodim20 (768×512)		 0.12 ms	 0.20 ms	 0.10 ms	 0.14 ms	 100 ms	 392 ms	 43 ms
 Small-Char (512×512)		 0.08 ms	 0.14 ms	 0.07 ms	 0.11 ms	 63 ms	 223 ms	 26 ms
 ISCV2_u2_v4 (8192×8192)		 10.8 ms	 34.9 ms	 12.3 ms	 12.3 ms	 14.1 s	 24.3 s	 2.7 s



QuickETC2: Fast ETC2 Texture Compression using Luma Differences ([nahjaeho.github.io](https://github.com/nahjaeho))





# 마무리

# 마무리

- 이번 시간에는 Assimp를 이용한 model loading 방법에 대해 살펴보았습니다.
  - Assimp의 개요
  - 이를 이용한 Mesh 및 Model 클래스
- 아울러 텍스처 압축의 개념과 방법에 대해서도 살펴 보았습니다.
  - 텍스처 압축 포맷의 특징과 대표 코덱들
  - BC1 및 ETC1의 원리

# 마무리

- 다음 실습 시간에는 아래 실습을 수행할 예정입니다.
  - Assimp 다운로드 및 빌드 [GitHub - assimp/assimp: The official Open-Asset-Importer-Library Repository. Loads 40+ 3D-file-formats into one unified and clean data structure.](#)
  - DLL 파일을 아래 모델 로딩 기반 코드에 연결하고, 다양한 모델을 읽어 들여 동작 테스트 [Code Viewer. Source code: src/3.model loading/1.model loading/model loading.cpp \(learnopengl.com\)](#)
  - Sponza scene을 읽어들이고 [https://casual-effects.com/g3d/data10/research/model/dabrovic\\_sponza/sponza.zip](https://casual-effects.com/g3d/data10/research/model/dabrovic_sponza/sponza.zip)
  - 여기에 flashlight 효과 추가 [Code Viewer. Source code: src/2.lighting/5.4.light casters spot soft/light casters spot soft.cpp \(learnopengl.com\)](#)