GIMS

Lab A2

In dieser Aufgabe müssen Sie einen Szenengraph zeichnen, der mittels der Bibliothek Assimp geladen wird. Es kann also Vorkommen, dass Sie gelegentlich Source-Code oder Dokumentation von Assimp lesen müssen!

Eine Klasse `Scene` hält die Daten, die zum Zeichnen auf der GPU benötigt werden. Die Factory Klasse `SceneGraphFactory` stellt die Schnittstelle zum Laden bereit.

Es empfiehlt sich als Ausgangspunkt die letzte Aufgabe zur Hand zu haben.

In dem Aufgabegerüst sind viele (`void`)`parameter` eingefügt. Entfernen Sie diese sobald Sie die Aufgabe bearbeiten!

Aufgabe 1 Root Signature

Bauen Sie eine für den Shader passende Root-Signature in `SceneGraphViewerApp::createRootSignature`!

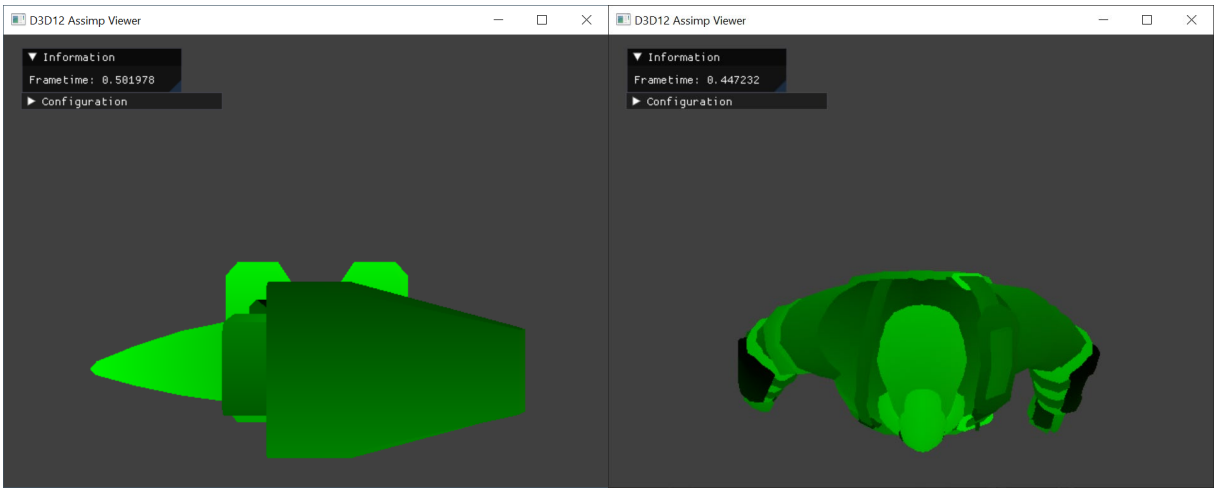
Aufgabe 2 Lab A1 abstrahieren – Teil 1.

Implementieren Sie die Methoden der Klassen `AABB`, `TriangleMeshD3D12`, `ConstantBufferD3D12` gemäß ihrer Kommentare.

Aufgabe 3 SceneGraphFactory::createMeshes

In `aiScene` befindet sich ein Array von Dreiecksnetzen `mMeshes`. Implementieren Sie `SceneGraphFactory::createMeshes` und `getTriangleIndicesFromAiMesh` um den Vektor `Scene::m_meshes` zu füllen.

Zeichnen Sie einzelne Netze. Als Farbe kann man zum Debuggen einfach die Texturkoordinaten nehmen. Mesh 0 und 1 sollten so aussehen:



Aufgabe 4 SceneGraphFactory::createNodes

Machen Sie sich mit `Scene::Node` vertraut. Sie stellt einen Knoten im Szenengraph dar. Alle Knoten werden flach in einem Vektor `Scene::m_nodes` gespeichert. Die Elemente von `Scene::m_nodes::childIndices` zeigen dabei auf Kind-Knoten, die sich innerhalb von `Scene::m_nodes` befinden.

In jedem Knoten speichern wir zusätzlich eine affine Transformation und einen Vektor von Mesh-Indizes `Scene::Node::meshIndices`. Die Elemente von `meshIndices` zeigen auf Elemente in `Scene::m_meshes`.

Vervollständigen Sie die Methode `SceneGraphFactory::createNodes`. Sie konvertiert den AssImp-Szenengraphen in eine für D3D12 darstellbare Form. Es wird bereits ein Wurzelknoten hinzugefügt. Von diesem aus soll mittels `createNodes` rekursiv der Vektor `Scene::m_nodes` befüllt werden und die Member-Variablen der Nodes `transformation`, `meshIndices` und `childIndices` entsprechend gesetzt werden!

Aufgabe 5 computeSceneAABB

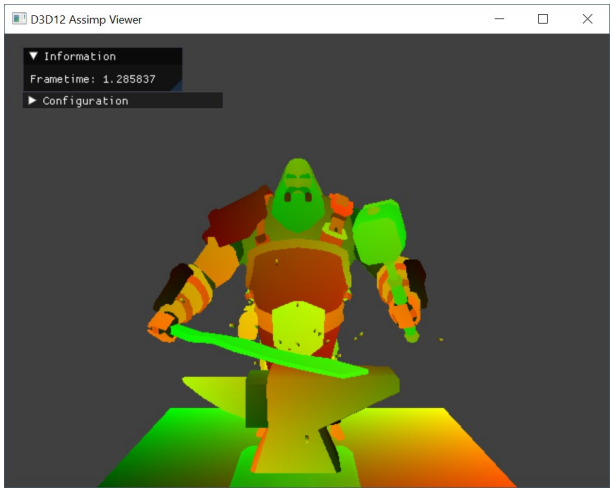
Berechnen Sie nun in `computeSceneAABB` die Axis-Aligned-Bounding-Box der gesamten Szene. Traversieren Sie dazu den Szenegraphen rekursiv und bilden Sie dabei die Vereinigungsmenge alle Mesh AABB. Beachten Sie, dass Sie die AABBs der Meshes richtig transformieren müssen.

Für den Nobel-Craftsman sollte folgende Werte für die Bounding-Box rauskommen:

Name	Value
m_lowerLeftBottom	-30.5716152, -2.56736803e-06, -43.0732880
m_upperRightTop	33.9077988, 11.0382118, 21.4061241

Aufgabe 6 Scene::addToCommandList

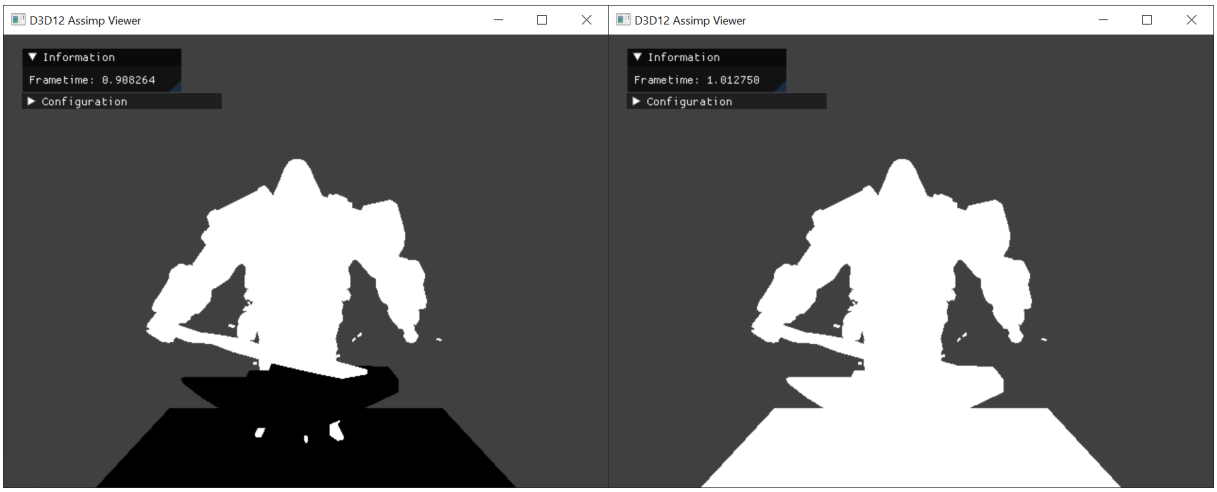
Traversieren Sie den Szenegraph, um die Meshes zu zeichnen. Implementieren Sie dazu `addToCommandListImpl`. Übergeben Sie die Transformation der Knoten mittels `SetGraphicsRoot32BitConstants`. Als Farbe kann man zum Debuggen einfach die Texturkoordinaten nehmen. Vergessen Sie nicht, die Szene zu normalisieren (z.B. in `SceneGraphViewerApp::drawScene`).



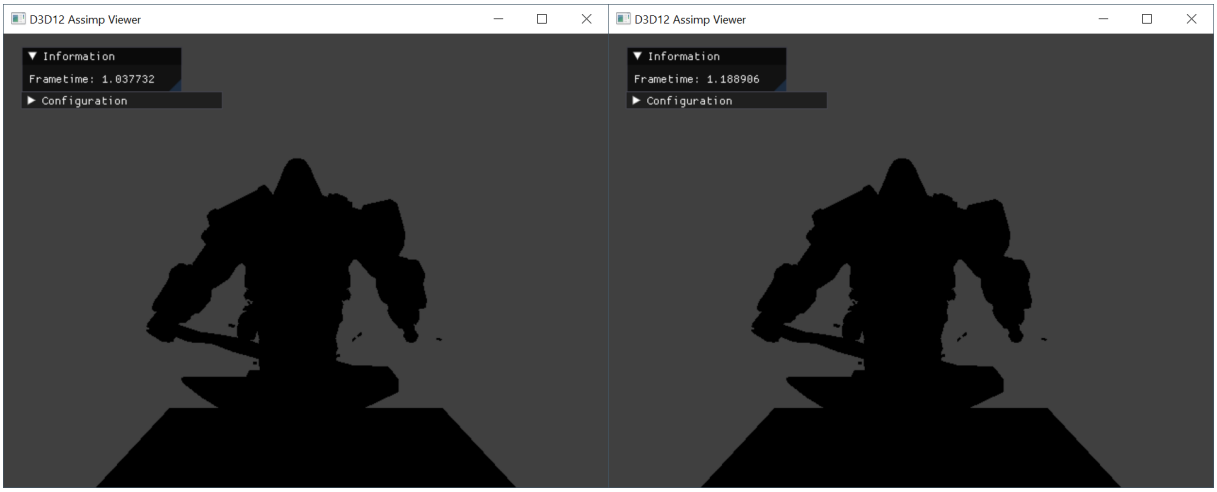
Aufgabe 7 Materialen erstellen und anwenden

Erzeugen Sie für jedes Material einen `Material::materialConstantBuffer` und befüllen Sie diesen geeignet in `SceneGraphFactory::createMaterials`. Lesen Sie die Materialen im Shader in `cbuffer Material : register(b2)` ein.

Ambient und Diffuse



Specular Color und Specular-Exponent



Aufgabe 8 Lab A1 abstrahieren – Teil 2.

Implementieren Sie die Methoden der Klasse `TextureD3D12` gemäß der Kommentare.

Aufgabe 9 Texturen erstellen

Erstellen Sie nun die Texturen für jedes Material in `SceneGraphFactory::createTextures`.

Falls bei einer Materialkomponente keine Textur vorliegt muss eine Default-1x1-Textur erstellt werden. Erstellen Sie dazu in `m_textures[0]`, `m_textures[1]` und `m_textures[2]` eine weiße, eine schwarze und eine blaue Textur (blau braucht man für Normal Maps!).

In der Hash-Map `textureFileNameToTextureIndex` finden Sie bereits eine Zuordnung von Textur-Filename auf Texturindex. Finden Sie heraus, wo diese Tabelle erstellt wird und analysieren Sie den Souce-Code.

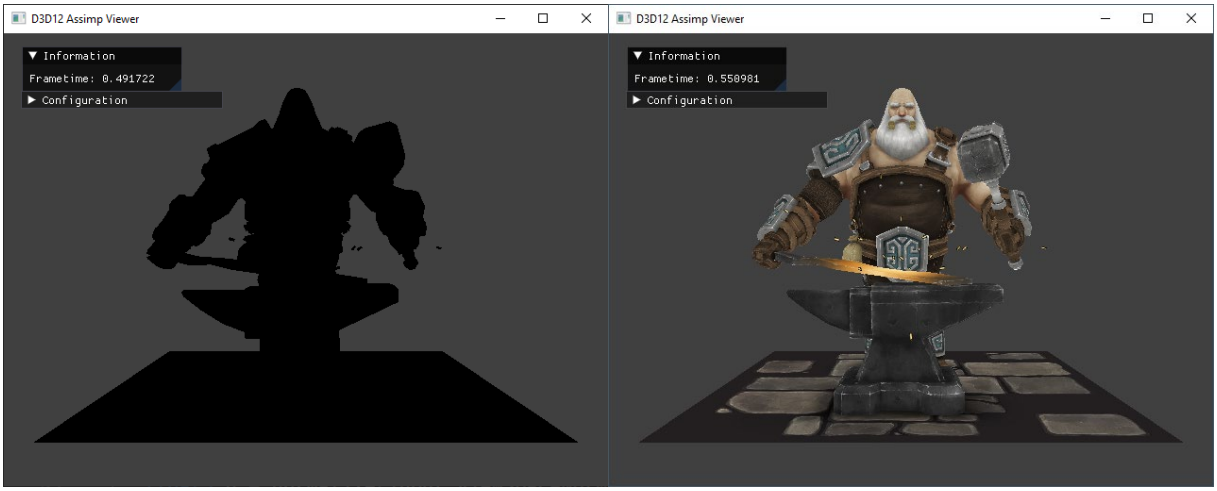
Erzeugen Sie anschließen die Texturen für jedes Material.

Aufgabe 10 Texturen anwenden

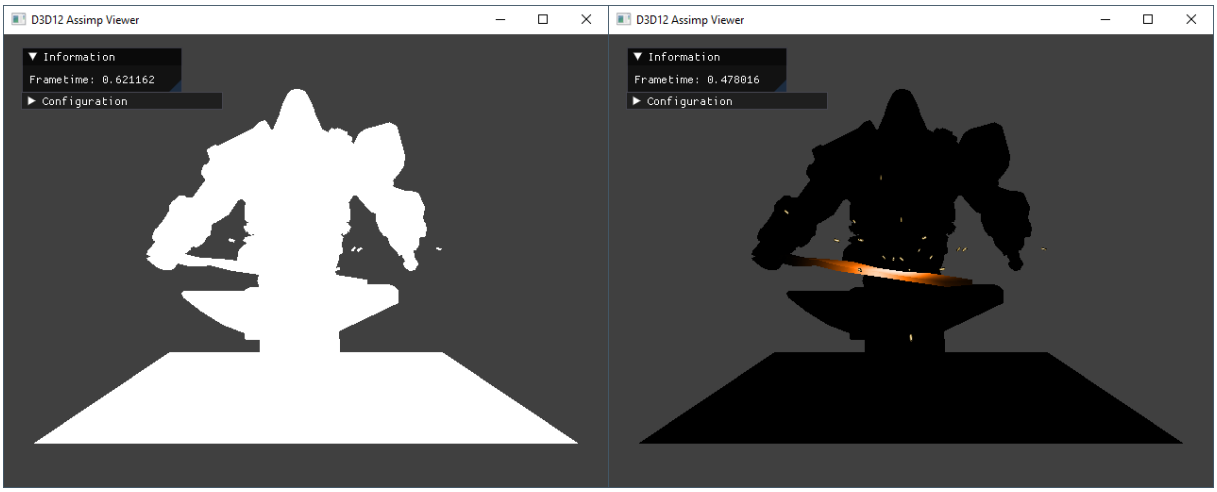
Modifizieren Sie nun `SceneGraphFactory::createMaterials` um für jedes Material die Texturen hinzuzufügen. Erzeugen Sie dazu einen Descriptor-Heap (`CreateDescriptorHeap`) mit fünf Descriptors und fügen Sie dem DescriptorHeap die Texturen hinzu. Nutzen Sie die Assimp Texturen `aiTextureType_AMBIENT`, `aiTextureType_DIFFUSE`, `aiTextureType_SPECULAR`, `aiTextureType_EMISSIVE` und `aiTextureType_HEIGHT` (als Normal Map!). Sollte keine geeignete Textur für das Material vorhanden sein, verwenden Sie eine passende Default-Textur!

Fügen Sie die Texturen in die Command-List (`commandList->SetDescriptorHeaps`, `SetGraphicsRootDescriptorTable`) und im Shader auf geeignete Weise hinzu. Samplen Sie zum Testen von jeder Textur! Sie sollten dann folgende Bilder erhalten.

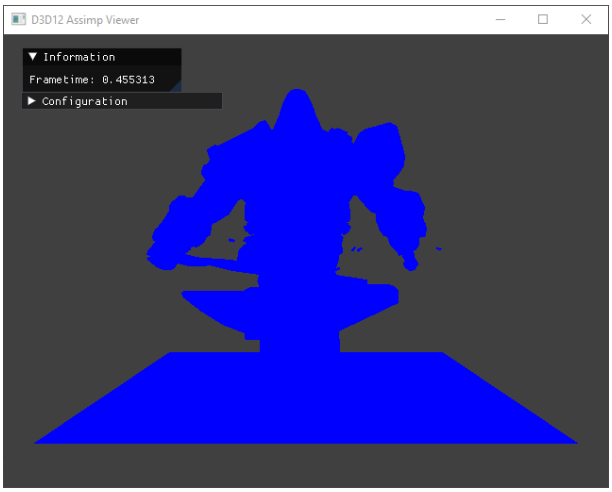
Ambient und Diffuse



Specular und Emissive

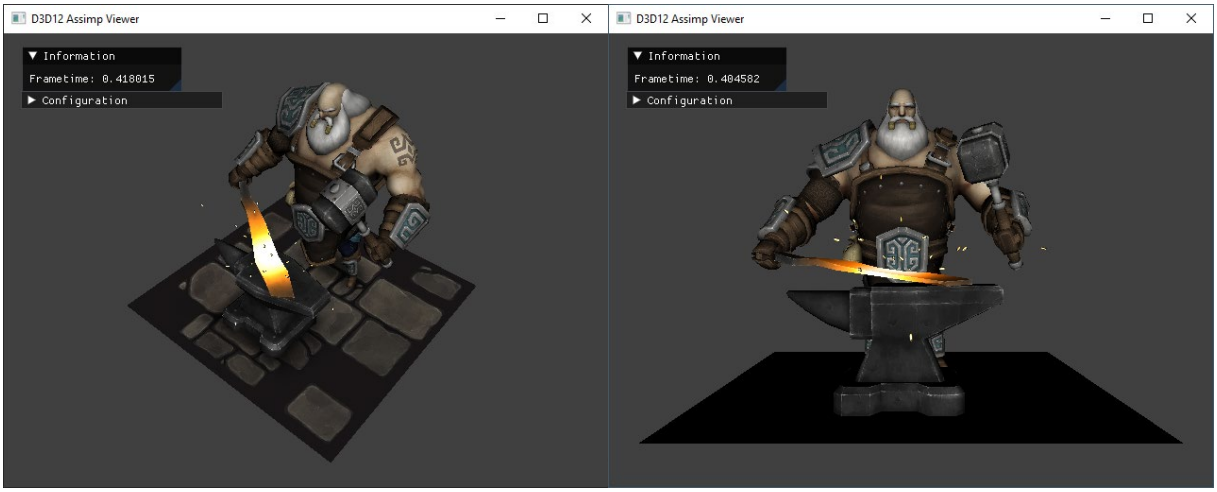


Normal Map



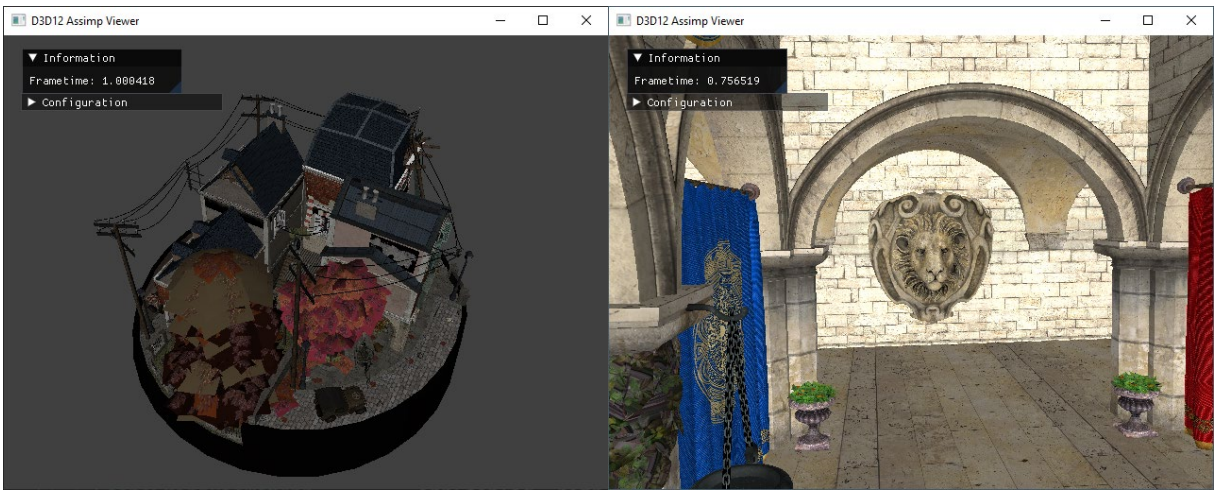
Aufgabe 11 Beleuchtung

Beleuchten Sie das Modell mit dem Blinn-Phong-Belechtungsmodell. Nutzen Sie für die Beleuchtungsanteile eine Kombination aus Texturwerten und den konstanten Werten.



Aufgabe 12 Andere Szene

In main.cpp können verschiedene Szene geladen werden. Probieren Sie die unterschiedliche Szenen aus. Laden Sie Szene von https://sketchfab.com/ in ihr Programm!



Aufgabe 13 Zusatzaufgaben

(a) Bauen Sie ein schickes DearImGui um Szenengraphen-Informationen in einem passenden UI-Element hierarchisch darstellen!
(b) Umhüllen Sie die einzelnen Objekte mit Bounding-Boxes!
(c) Positionieren Sie die Kamera beim Laden des Objektes so, dass die ganze Szene zu sehen ist!
(d) Lassen Sie Position und Farbe der Lichtquelle per UI Steuern.
(e) Platzieren Sie mehrere Lichtquellen in der Szene!