Tools Programming

Asset integration pipeline 3

Class 8

Index

- Engine importer
- Utility tools
- Custom Components
  - Export
  - Import
- Prefabs
- Refining and optimization
  - Exporting mesh data to binary
  - Refining the UI
- Engine interface
  - ImGUI
  - Building the hierarchy

Engine importing

- Finish the scene importer in the engine
- In Parsers.cpp implement the following logic:
  - Finish entity parsing on ParseScene function
  - Finish entity parsing components on ParseEntity method:
    - Parse prefabs
    - Parse transform component
    - Parse render component
    - Parse light
  - Generate the given entity in the world with all his listed components

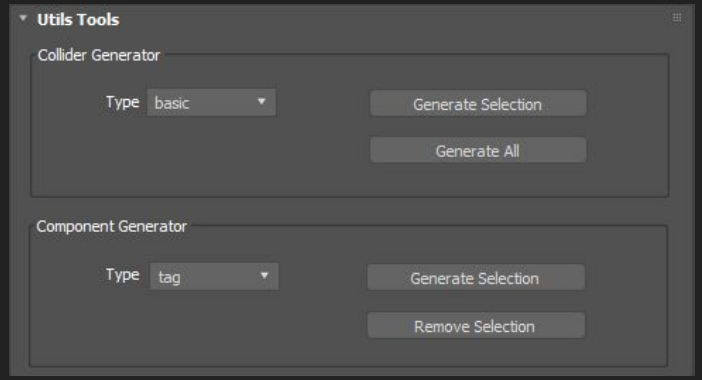
- Remember to load an scene on game.cpp with the following line
  - Parsers::parseScene("data/assets/scenes/scene_test.scene", graphics_system_);

Utility tools

- Set of tools to aid us within 3DSMax to help us on generating scene data

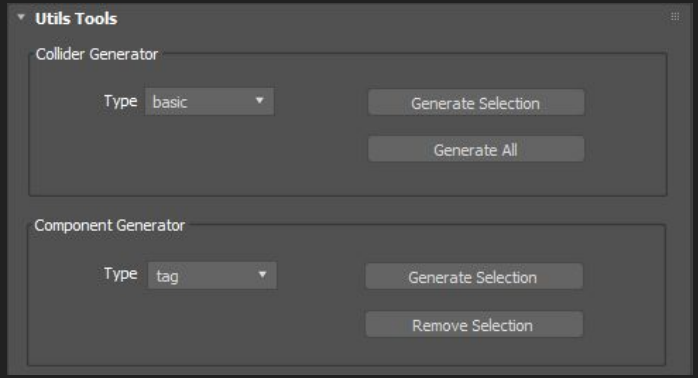
- Collider generator: Used to generate three types of possible colliders
  - Box collider
  - Convex collider
  - Mesh collider

- Component generator: Used to add components to the selected object in the scene.
  - Data is permanently saved
  - User should be able to remove the component attached.



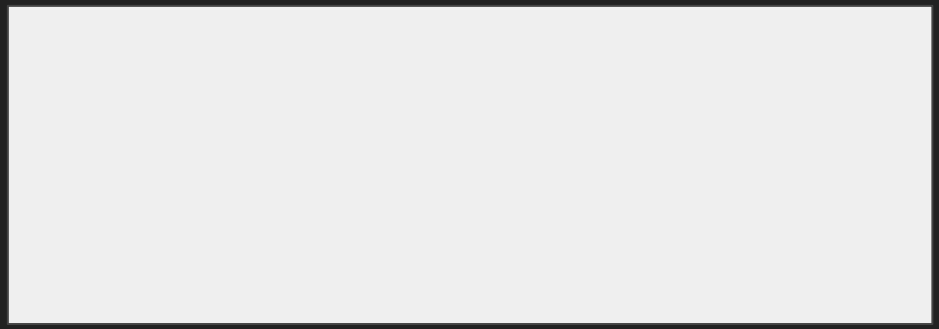
Sample

- Update the UI to provide exporting tools for:
  - Component generator
  - Collider generator
- Export can be done with:
  - All scene items
  - Only selected items.
- Components types can be: custom classes such as tag, collider
- Colliders types can be: box, convex, mesh



Custom Components

- Components that are not basic (e.g transform,light,prefab…)
- Components are created by using the attributes node from MaxScript
- Component attributes have the following sections:
  - Parameters section: All attributes of the given type
  - Rollout section: UI to be displayed in the Command panel+
- Custom components functionality
- Implement tag custom component in the mvd_components file



Sample Component

- Component in Maxscript is represented by attributes
  - Composed by parameters, defining its attributes
  - Rollout used to display the options within the UI

```
parameters cmp_col rollout:cmp_col
(
	col_group type: #string ui:col_group default:"All"
	col_mask type: #string ui:col_mask default:""
	is_trigger type: #boolean ui:trigger default:false
	is_dynamic type: #boolean ui:dynamic default:false
	is_controller type: #boolean ui:controller default:false
	is_gravity type: #boolean ui:isgravity default:false
)
```

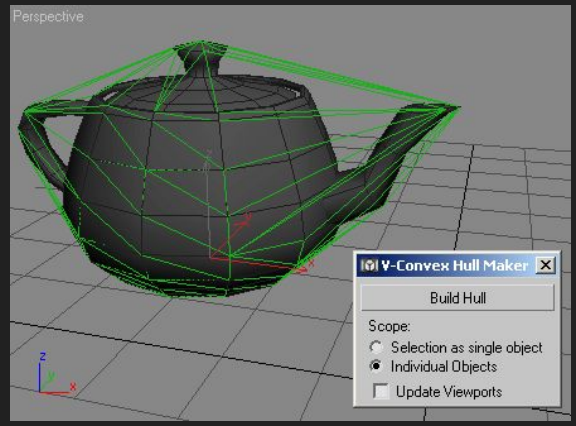
```
rollout cmp_col "Component Collider" width:162 height:174
(
		edittext 'col_group' "Group" width:130 height:20 align:#center
		edittext 'col_mask' "Mask" width:130 height:20 align:#center
		checkbox 'trigger' "Is Trigger" width:70 height:15
type:#BOOLEAN align:#center
		checkbox 'dynamic' "Is Dynamic" width:76 height:15
type:#BOOLEAN align:#center
		checkbox 'controller' "Is Controller" width:83 height:15
type:#BOOLEAN align:#center
		checkbox 'isgravity' "Is Gravity" width:71 height:15
type:#BOOLEAN align:#center

)
```

Collider generator

The collider generator must generate:

- Box collider: Bounding volume of the object
  - Can be obtained by bounding volume max func
- Convex collider: Convex volume
  - Can be obtained using the nvpx.CreateConvexHull function built-in 3DSMax
- Mesh collider: Copy volume:
  - Can be obtained by copying the object the same it is
  - Modifications can be done afterwards.



Sample

- Implement collider generation for the types of colliders
  - Box collider: simple bounding box involving the object
  - Convex collider: point cloud of vertices that form a non concave object enclosing the original source object
  - Mesh collider: an identical collision representation of the mesh.

- Once a collider MUST:
  - Be a children of the parent mesh
  - Be in colliders layer

Prefabs

- Prefabs are pre-sets of objects used to easily multiplicate them and place them in an scene
- The key concept is Instancing. Instancing allow us to
  - REUSE the given data in the engine (Performance)
  - CONTROL all the instances under the root prefab (Chain)
  - ...

Sample

- Generate a prefab as XRef and place it around your scene
- Read the scene and detect whether an object is an xref or a simple model
- Export the prefab within the scene file
- Import the prefab as a prefab in the engine. (reading another file)

TO-DO

Export mesh to binary

- OBJ is a very slow format
- We need to work with something that is almost instant to read by the computer.

Computer understands BYTES

- Compress the mesh data into binary information.
- Binary information needs an standard codification to be able to read it later
- Many formats use this codification (e.g TGA)

Link: https://en.wikipedia.org/wiki/Truevision_TGA

Export mesh to binary

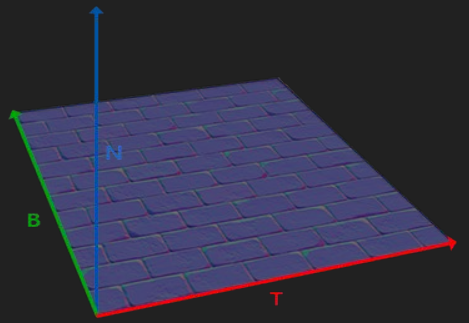
- The binary format must have:
  - A header sequence to identify the:
    - Number of vertices
    - Number of faces
    - Number of materials
    - Size of the previous elements
  - Vertices, faces, normals and materials data in local space.
  - Normals must be exported in tangent space.
- Modify Maxscript exporter to export mesh data into binary file.
- Modify mesh importer to import mesh data from binary file in the engine

TO-DO

Export mesh to binary

- Normal vectors are expressed in tangent space
- Space that is local to the surface of the triangle.
- To do so we need to calculate what is known as TBN Matrix, using tangent bitangent and normal vectors.

$$\vec{N} = \text{normal}$$
$$\vec{T} = \text{tangent}$$
$$\vec{B} = \vec{N} \times \vec{T}$$
$$TBN = \begin{bmatrix} \vec{T}.x & \vec{T}.y & \vec{T}.z \\ \vec{B}.x & \vec{B}.y & \vec{B}.z \\ \vec{N}.x & \vec{N}.y & \vec{N}.z \end{bmatrix}$$



Link: http://foundationsofgameenginedev.com/FGED2-sample.pdf
Link: https://learnopengl.com/Advanced-Lighting/Normal-Mapping