CS11: 3D Scanning of Large Spaces

2/26/2021 Meeting

Brandon Withington, Seika Mahmud, Casey Boomer

Metashape Plugins :
Metashape uses python libraries to integrate it with plugins. They can be accessed through the scripts folder in the Metashape file. Here is a link to the documentation:
metashape_python_api_1_5_0.pdf (agisoft.com)
This library is extremely useful for us, and would require very little coding to get what we would want from it.

There is also an option to not have to feed the plugin through the application, which would suit our interests very well. However, there are current issues with getting modules installed properly to use the Metashape libraries.

Functions to use:

- importPoints(path[, format][, projection][, shift][, progress])
    - Import point cloud from file. Parameters • path (string) – Path to point cloud. • format (PointsFormat) – Point cloud format. • projection (CoordinateSystem) – Point cloud coordinate system. • shift (3-element vector) – Optional shift to be applied to point coordinates. • progress (Callable[[float], None]) – Progress callback.
- resetRegion()
    - Reset reconstruction volume selector to default position
- buildModel(surface=Arbitrary, interpolation=EnabledInterpolation, face_count=MediumFaceCount[, source][, classes], vertex_colors=True, quality=MediumQuality, volumetric_masks=False, keep_depth=False, reuse_depth=False[, progress])
    - Generate model for the chunk frame. Parameters • surface (SurfaceType) – Type of object to be reconstructed. • interpolation (Interpolation) – Interpolation mode. • face_count (FaceCount or int) – Target face count. • source (DataSource) – Selects between dense point cloud, sparse point cloud and depth maps. If not specified, uses dense cloud if available. • classes (list of PointClass) – List of dense point classes to be used for surface extraction. • vertex_colors (bool) – Enables/disables vertex colors calculation. • quality (Quality) – Depth map quality. Ignored if source is not DepthMapsData. • volumetric_masks (bool) – Enables/disables strict volumetric masking. • keep_depth (bool) – Enables keep

depth maps option. • reuse_depth (bool) – Enables reuse depth maps option. • progress (Callable[[float], None]) – Progress callback.

- buildTexture(blending=MosaicBlending, size=2048, fill_holes=True, ghosting_filter=True[, cameras][, progress])
    - Generate texture for the chunk. Parameters • blending (BlendingMode) – Texture blending mode. • size (int) – Texture size. • fill_holes (bool) – Enables hole filling. • ghosting_filter (bool) – Enables ghosting filter. • cameras (list of Camera) – A list of cameras to be used for texturing. • progress (Callable[[float], None]) – Progress callback.
- exportModel(path, binary=True, precision=6, texture_format=ImageFormatJPEG, texture=True, normals=True, colors=True, cameras=True, markers=True, udim=False, alpha=False, strip_extensions=False, raster_transform=RasterTransformNone, colors_rgb_8bit=True[, comment][, format][, projection][, shift][, progress])
    - Export generated model for the chunk. Parameters • path (string) – Path to output model. • binary (bool) – Enables/disables binary encoding (if supported by format). • precision (int) – Number of digits after the decimal point (for text formats). • texture_format (ImageFormat) – Texture format. • texture (bool) – Enables/disables texture export. • normals (bool) – Enables/disables export of vertex normals. • colors (bool) – Enables/disables export of vertex colors. • cameras (bool) – Enables/disables camera export. • markers (bool) – Enables/disables marker export. • udim (bool) – Enables/disables UDIM texture layout. • alpha (bool) – Enables/disables alpha channel export. • strip_extensions (bool) – Strips camera label extensions during export. • raster_transform (RasterTransformType) – Raster band transformation. • colors_rgb_8bit (bool) – Convert colors to 8 bit RGB. • comment (string) – Optional comment (if supported by selected format). • format (ModelFormat) – Export format. • projection (CoordinateSystem) – Output coordinate system. • shift (3-element vector) – Optional shift to be applied to vertex coordinates. • progress (Callable[[float], None]) – Progress callback.

These functions listed cover every process that we use in the actual application. The long parameters for the functions would be defined by the settings that we would have in the user interface.

Currently, the roadblock we have with these plugins is currently the problems with importing the Metashape modules. Once

Metashape Prices :

When we were testing out metashape, we noted that it should be possible for us to use the standard version of it which is significantly cheaper than the pro version. However, we were incorrect as the pro version is the only version that supports the api and scripting functionality. This may be an issue, as the professional package costs $3499 for non educational licenses.

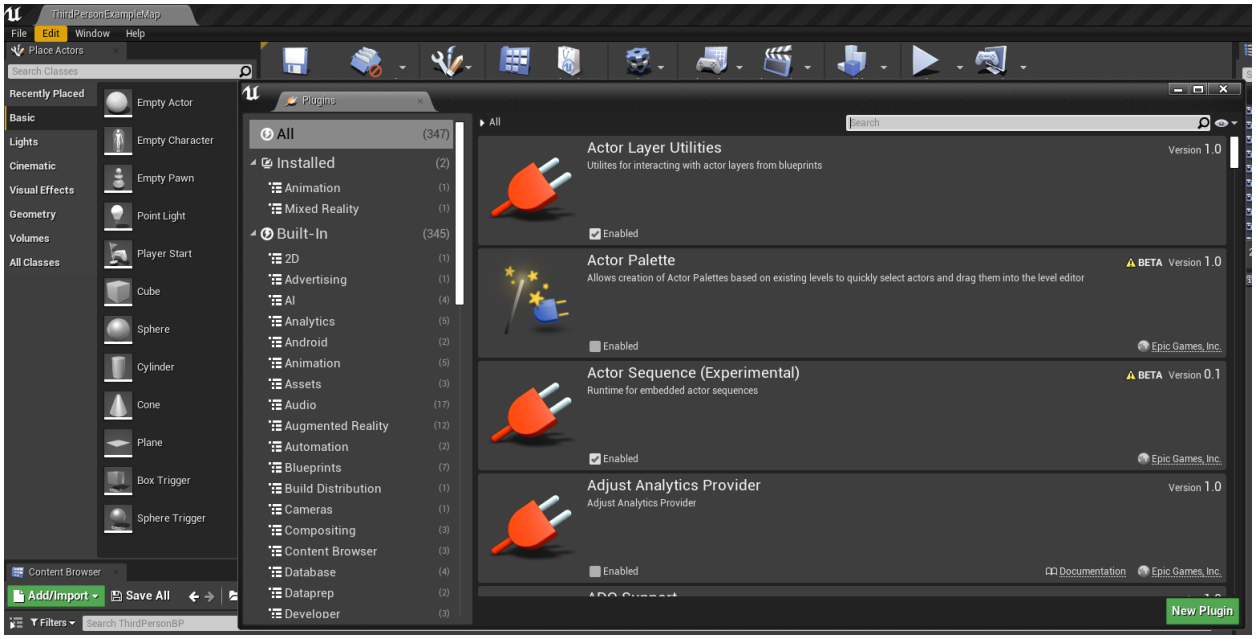
This is difficult because we are so far into the project, we need to start working on a software that we can use within our scope. Here are some options:

- We continue to use Metashape, since the educational version of the pro license is $549. Educators would be able to access this lower price, and the cost can be further lowered by purchasing a pack of multiple licenses. Since virtual classrooms are one of the major talking points for our project, this could work. This large fee is still only a one time fee; with Matterport, the user is required to purchase the camera for $3000, then they are required to pay a monthly subscription of either $69 a month, or $369 a month. A business could see using our software as significantly cheaper due to the monthly fees adding up(Dot 3D has a yearly fee, but it is much less than the matterport one. Furthermore, since our interface is focused mostly on the second half of the pipeline now, the user could choose to forego Dot3D in favor of a cheaper or even free software.
- We start looking for another software to use from the given list from chris. We would likely need to rush to get a software that is compatible with us. Alternatively, we could stick with Metashape for now, but keep looking for software that could be cheaper and integrate it into what we wrote for Metashape.

Unreal Engine Plugins :

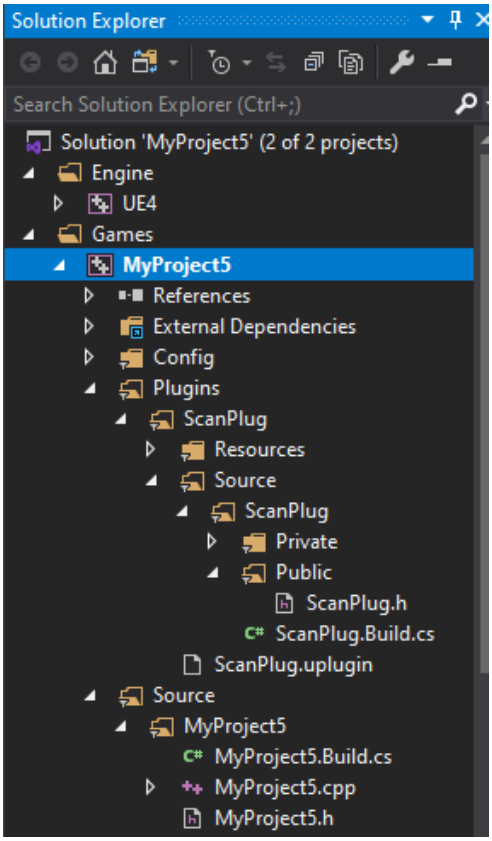
Creating plugins for the Unreal Engine is an easy process that can be started in the first menu of the creation of a project, you need to make sure that you have downloaded Microsoft Visual Studio. These settings will enable you to create plugins :

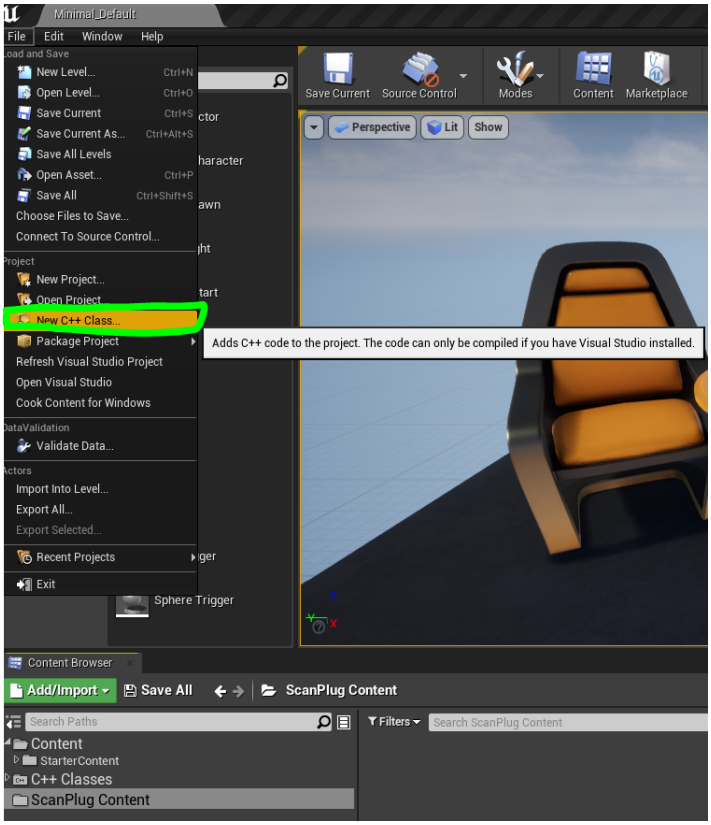
Then you can access the plugin panel through the edit sub-menu of the Unreal Engine :



Once in this menu you can start creating a new plugin by clicking on the new plugin button, there are many different templates that you can start off with but it is recommended to go with the blank template.

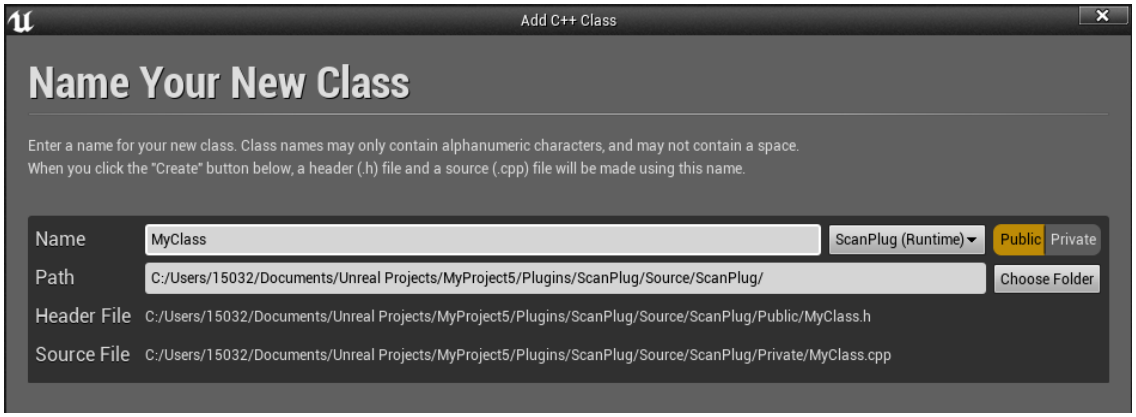
You can confirm whether or not a plugin has been made by checking visual studio's solution explorer. Once that is done, you need to create a new c++ class for the plugin, you need to create this in the same folder as your plugin folder you generated.



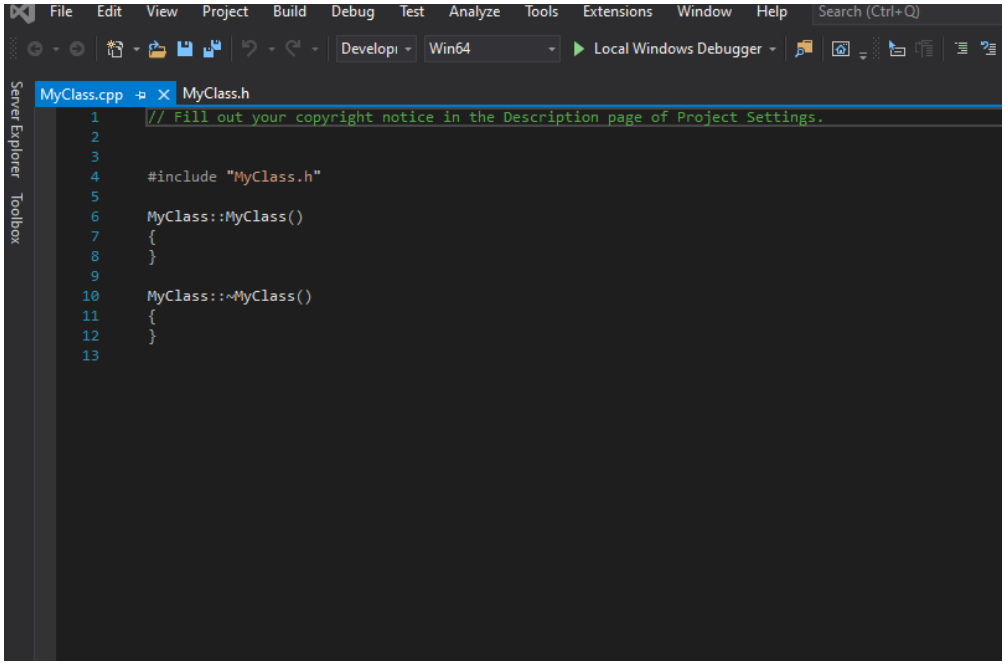


Note that there are other Unreal Engine plugins that enable the use of python. For our purposes those might be very useful. Otherwise, we would have to begin plugin-creation with C++.

When you create your new C++ class you need to ensure that it is being generated into the plugin you previously made.



Now you can start developing the plugin!



To start scripting in python, refer to : [Scripting the Editor using Python | Unreal Engine Documentation]