|  |  |  |
| --- | --- | --- |
|  | **Rochester Institute of Technology**  **Golisano College of Computing and Information Sciences**  **School of Interactive Games and Media**  **2145 Golisano Hall – (585) 475-7680** |  |

**Data Structures & Algorithms for Games & Simulation II**

**IGME 309, 2016-17 spring**

**E07: LERP**

For this ICE I will suggest you to use ReEngine, you may use your own solution if you want as well. There is NO start code for this ICE but you may use the Sandbox project as startup and it is available at: <https://github.com/labigm/ReEngineApp_2016s> under **00\_Sandbox** project.

This project will generate a random number of spheres in increasing color.

In order to use it, you will need to:

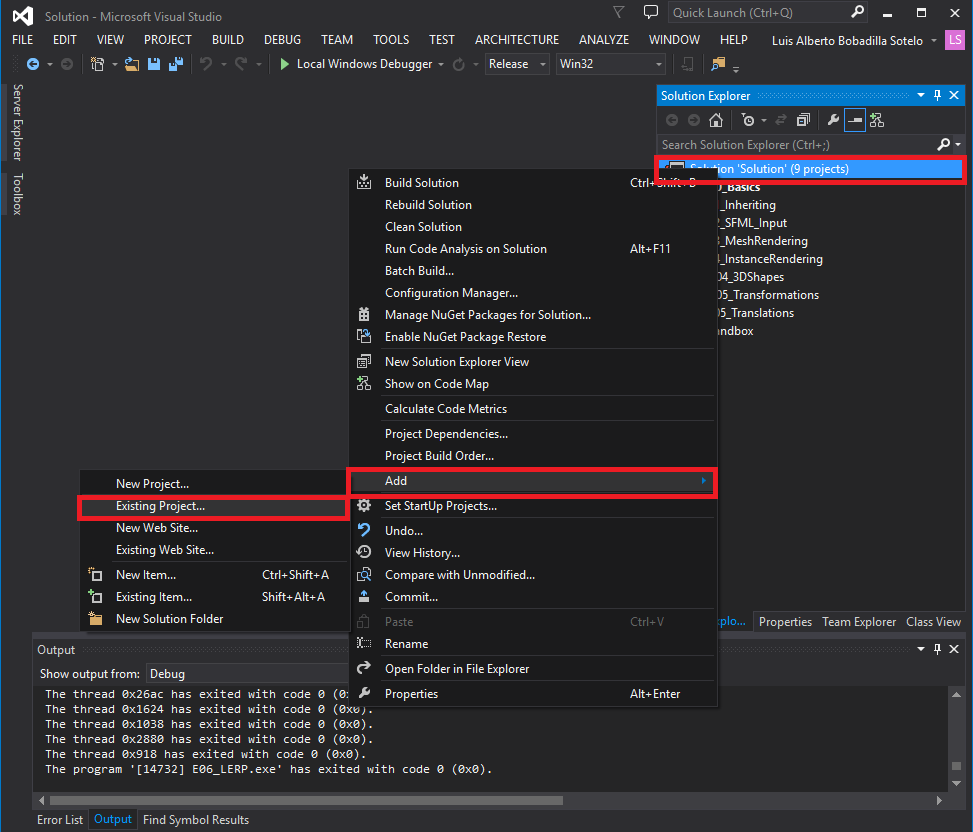
1. Duplicate the project under windows (copy paste it in the same folder)
2. Rename it to E07\_Lerp
3. Rename the following files to E07\_Lerp as well
   1. 00\_Sandbox.vcxproj 🡪 E07\_Lerp.vcxproj
   2. 00\_Sandbox.vcxproj.filters 🡪 E07\_Lerp.vcxproj.filters
   3. 00\_Sandbox.vcxproj. user 🡪 E07\_Lerp.vcxproj. user
4. Edit E07\_Lerp.vcxproj in notepad++ (or any other text editor) and change the following line:

<ProjectName>Sandbox</ProjectName>

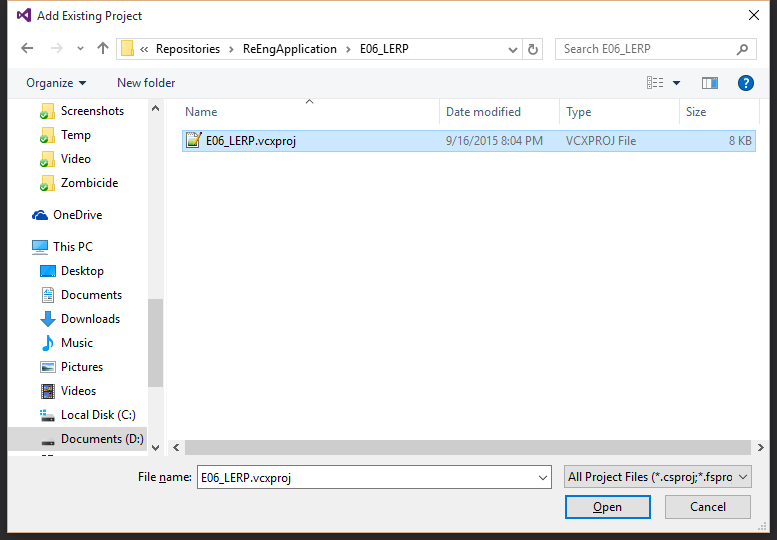
To

<ProjectName>E07\_Lerp</ProjectName>

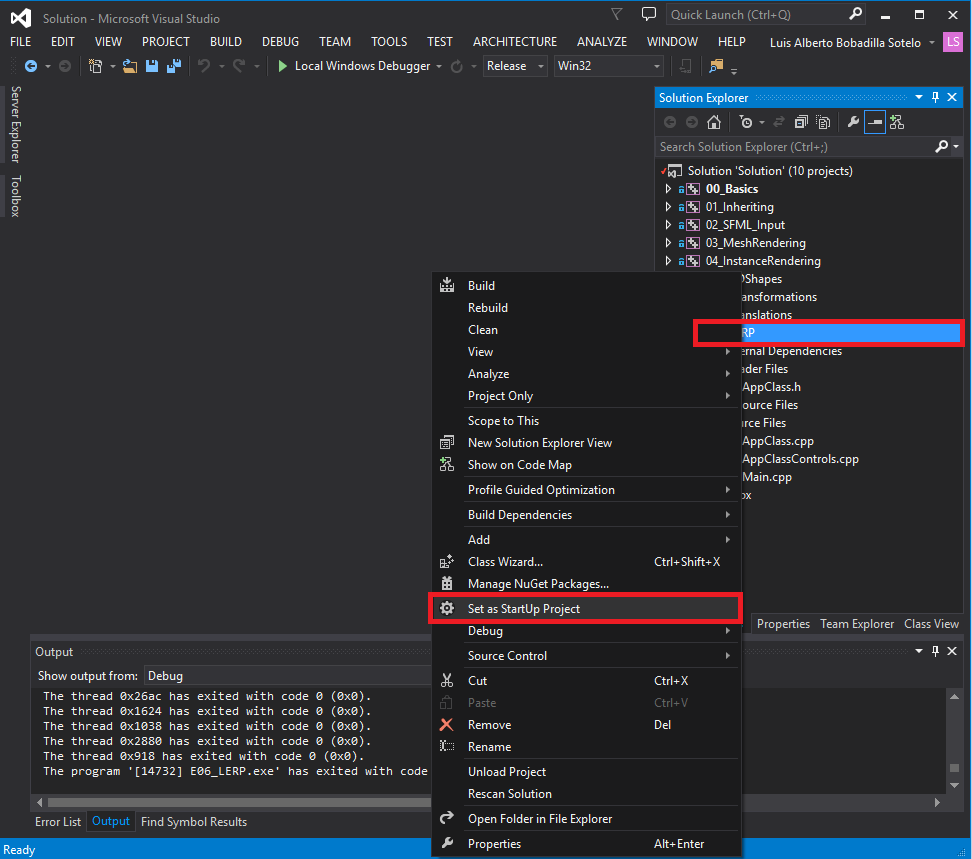
1. Right-click on the project solution and add an existing project



1. Browse for the file E07\_Lerp.vcxproj and add it.



1. Right-click on the project and set it up as the StartUp project:



1. Making sure you are in the AppClass.h of E07\_Lerp add the following variables, they will count the number of objects on the scene, will hold their info and their matrices.
   1. int m\_nObjects;
   2. PrimitiveClass\* m\_pSphere = nullptr;
   3. matrix4\* m\_pMatrix = nullptr;
2. in InitVariables set a new camera position:

m\_pCameraMngr->SetPosition(vector3(0.0f, 0.0f, 35.0f));

1. Initialize the m\_nObjects variable to a random number using rand():

srand(time(NULL));

m\_nObjects = rand() % 23 + 5;

1. Make a couple of start and end vectors like so:

vector3 v3Start = vector3(-m\_nObjects, 0.0f, 0.0f);

vector3 v3End = vector3(m\_nObjects, 0.0f, 0.0f);

1. Initialize the m\_pSphere variable and the m\_pMatrix variable to a new set of appropriate variables using m\_nObjects as the quantity as in a list of pointers:

ObjectPointer = new ObjectType[quantity];

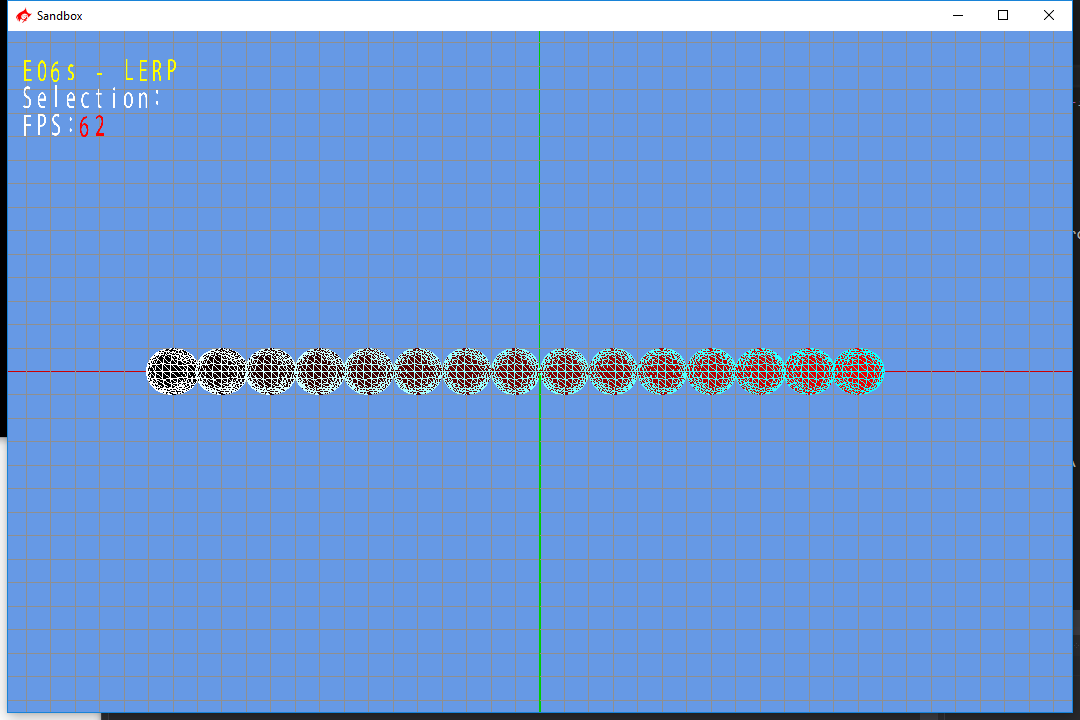
1. In a **for loop** that goes from the first to the last sphere add a MapValue function call that will generate a percentage of the lerp like this:

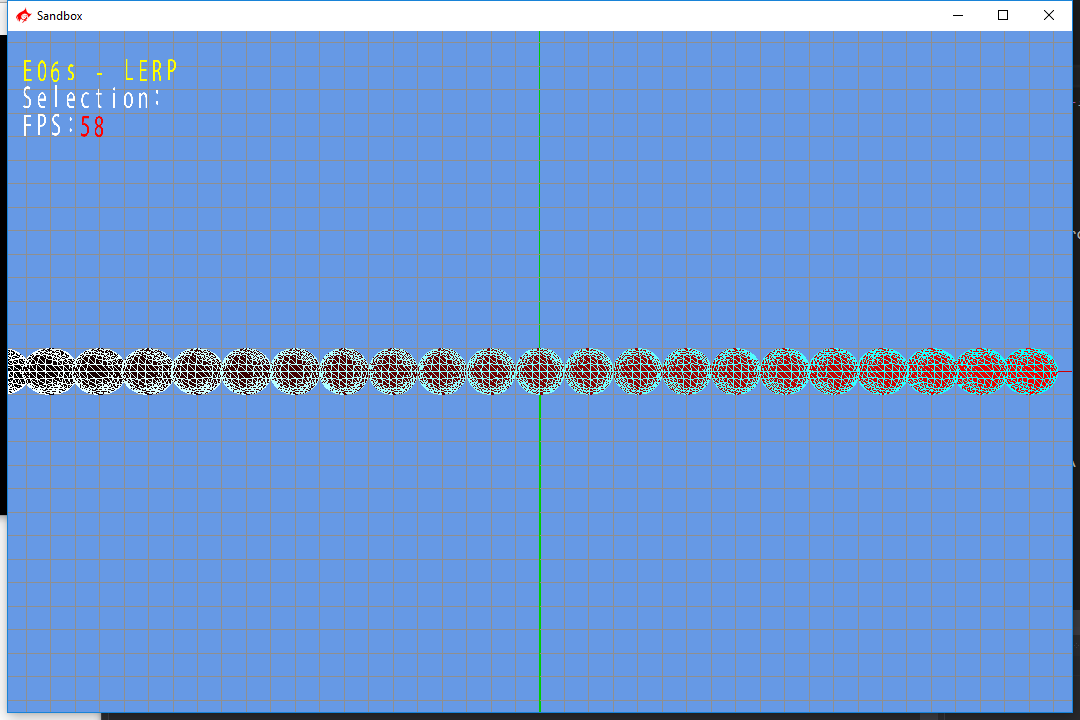
float fPercent = MapValue(static\_cast<float>(i), 0.0f, static\_cast<float>(m\_nObjects), 0.0f, 1.0f);

A more in depth description of the MapValue method can be found at the end of this document.

1. Inside of the same for loop generate every Primitive as a new sphere of radius 1, with a subdivision level of 5 and a new vector3(fPercent, 0.0f, 0.0f) as the color.
2. Inside of the same for loop update the position of each matrix to a translated vector of the return of the linear interpolation from v3Start and v3End according to the fPercent
3. In Display render all of the spheres (using their respective matrices)
4. In Release deallocate the memory you reserved for m\_pSphere and m\_pMatrix remember that they reserved memory as an array []

If everything went according to plan you should have a program like the one exemplified under \_Binary as E07\_Lerp DEMO that will generate a random number of spheres each time:





Show this to the professor or TA and submit to the dropbox labeled E07 Lerp.

For your submission you only need to zip the folder E07 – LERP, nothing else is required, and the file size should not be larger than 100kb. Also remember to add your repository address to your submission comment and push your changes to it.

Information on the MapValue Function:

This function is a copy of the map function of the Processing language, it will transform a value from an original scale to a new scale, for example you have a scale that goes from 0 to 10 originally but you need a value on a scale that goes between 0 and 100 a value that originally is worth 5 in the new scale it would be worth 50. The definition is as follows:

/\*

MapValue

USAGE: Will map a value from an original scale to a new scale

ARGUMENTS:

T valueToMap -> input value

T originalScale\_min -> Start of the original scale

T originalScale\_max -> End of the original scale

T mappedScale\_min -> Start of the new scale

T mappedScale\_max -> end of the new scale

OUTPUT: returns the mapped value

\*/

template <class T>

static T MapValue(T valueToMap, T originalScale\_min, T originalScale\_max, T mappedScale\_min, T mappedScale\_max)

{

return (valueToMap - originalScale\_min) \* (mappedScale\_max - mappedScale\_min) / (originalScale\_max - originalScale\_min) + mappedScale\_min;

}

}