Virtual world application development
for immersive 3D data visualization
with the Oculus Rift

Johann Mitloehner, 2016/17

Immersive data visualization and virtual data exploration have become available for average users with the introduction of affordable VR goggles like the Oculus Rift. The design of meaningful applications other than games for this relatively new type of hardware provide both **potentials and challenges in abundance**. This thesis topic is about **developing a basic virtual reality application** using the Rift hardware for some type of data visualization or exploration; details to be discussed with supervisor.

- Workstation with an **Oculus Rift and the required PC** hardware are available
- **Programming skills** are required: **C++**
- However, we are not looking for finished products but rather **proof-of-concept** to demonstrate the unique advantages of immersive 3D virtual worlds for data visualization and exploration.
- Rift goggles are used for testing and evaluation, but **most of the development can be done on a conventional PC**.

References:

Drouhard, Margaret, et al. Immersive Visualization for Materials Science Data Analysis using the Oculus Rift. In: 2015 IEEE International Conference on Big Data. IEEE, 2015, p. 2453-2461.

Donalek, Ciro, et al. Immersive and collaborative data visualization using virtual reality platforms. In: 2014 IEEE International Conference on Big Data. IEEE, 2014, p. 609-614.

Marks, Stefan, Javier E. Estevez, and Andy M. Connor. Towards the Holodeck: fully immersive virtual reality visualisation of scientific and engineering data. In: Proceedings of the 29th International Conference on Image and Vision Computing New Zealand. ACM, 2014.

Sample Code:

```
#include "MyProject.h"
#include "FloatingActor.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include <stdlib.h>
using namespace std;

AFloatingActor::AFloatingActor()
{
	PrimaryActorTick.bCanEverTick = true;
	bHidden = false;
```

```
	USphereComponent* SphereComponent =
CreateDefaultSubobject<USphereComponent>(TEXT("RootComponent"));
	RootComponent = SphereComponent;
	SphereComponent->InitSphereRadius(40.0);
	SphereComponent->SetCollisionProfileName(TEXT("Pawn"));
	UStaticMeshComponent* SphereVisual =
CreateDefaultSubobject<UStaticMeshComponent>(TEXT("VisualRepresentation"));
	SphereVisual->SetupAttachment(RootComponent);
	static ConstructorHelpers::FObjectFinder<UStaticMesh>
SphereVisualAsset(TEXT("/Game/StarterContent/Shapes/Shape_Sphere.Shape_Sphere"));
	if (SphereVisualAsset.Succeeded())
	{
		SphereVisual->SetStaticMesh(SphereVisualAsset.Object);
		SphereVisual->SetRelativeLocation(FVector(0.0f, 0.0f, -40.0f));
		SphereVisual->SetWorldScale3D(FVector(0.8));
	}
	static ConstructorHelpers::FObjectFinder<UMaterial>
Material(TEXT("MaterialInstanceConstant'/Game/StarterContent/Materials/M_Metal_Burnished_S
teel.M_Metal_Burnished_Steel'"));
	if (Material.Succeeded()) {
		SphereVisual->SetMaterial(0, Material.Object);
	}

}

// Called when the game starts or when spawned
void AFloatingActor::BeginPlay()
{
	Super::BeginPlay();
	UE_LOG(LogTemp, Warning, TEXT("BeginPlay: Actor %s at %s"), *this-
>GetFName().ToString(), *this->GetActorLocation().ToString());

	ifstream myfile;
	myfile.open("d:\\data.txt");
	myfile >> ScaleFactor;
	myfile.close();
}

// Called every frame
void AFloatingActor::Tick(float DeltaTime)
{
	Super::Tick(DeltaTime);
	FVector NewLocation = GetActorLocation();
	float DeltaHeight = (FMath::Sin(RunningTime + DeltaTime) - FMath::Sin(RunningTime));
	int i;
	NewLocation.Z += DeltaHeight * (float)ScaleFactor; // 20.0f;	//Scale our height by a
factor of 20
	RunningTime += DeltaTime;
	SetActorLocation(NewLocation);
	if (Iteration > 0)
	{
		ReadCSV();
```

```
		for (i = 0; i < row; i++) {
			// Make a location for the new actor to spawn at (300 units above this actor)
			FVector SpawnLoc = GetActorLocation() + FVector(array[i][0], array[i][1],
900 + array[i][2]); //  GetActorLocation() + FVector(0.f, 50 + i * 50, 0);

			scale = 0.8; // array[i][0];
			// Spawn the new actor
			FActorSpawnParameters SpawnParams;
			SpawnParams.Owner = this;
			SpawnParams.Instigator = Instigator;
			AFloatingActor* NewActor = GetWorld()-
>SpawnActor<AFloatingActor>(AFloatingActor::StaticClass(), SpawnLoc, FRotator::ZeroRotator,
SpawnParams);
			NewActor->SetActorScale3D(FVector(scale));
			NewActor->Iteration = 0; //  so that we dont spawn new actors forever
			UE_LOG(LogTemp, Warning, TEXT("Tick: SpawnActor %s at %s"),
*NewActor->GetFName().ToString(), *NewActor->GetActorLocation().ToString());
		}
		Iteration = 0; // stop spawning

	}
}

void AFloatingActor::ReadCSV() {
	std::ifstream file("d:\\miser.csv");
	std::string line;
	col = 0;
	row = 0;
	while (std::getline(file, line))
	{
		std::istringstream iss(line);
		std::string result;
		while (std::getline(iss, result, ','))
		{
			array[row][col] = atof(result.c_str());
			UE_LOG(LogTemp, Warning, TEXT("row %d col %d data %f"), row, col,
array[row][col]);
			col = col + 1;
		}
		row = row + 1;
		col = 0;
	}
}
```