Ankoor Shah
Final Report – cs493
amshah4

Research (Group/Individual)

When the project first began, all progress was primarily in terms of research. As a part of the team working on the Unreal Engine, the main problem was developing the Unreal Engine mindset, which we addressed by simple experimentation and making small changes and adjustments to default projects and observing the effects. The Unreal Engine contains a variety of default project, but most experimentation was done with the First-Person project provided, since we knew Oculus Rift support was a major goal for our immersive environment. Manipulating this basic Unreal Engine 4 project was our first challenge. Unreal Engine 4 has no built in scripting mechanism; all data manipulation withing the Engine is done by either using what are called Blueprints, or by manipulating the project C++ source, which can only be done with Visual Studios Express 2013 or later.

Learning about the Unreal Blueprint system was done primarily from the Unreal Engine wiki, Unreal Engine AnswerHub, and their youtube channel, both of which are cited below. Understanding this enormous and sophisticated system proved to be extremely difficult, despite the huge pool of resources, because the smallest changes in the settings would have a chaotic effect on the environment, and the effects were not very modular. Disabling collision for certain objects might disable gravity. Many times, these kinds of effects had to do with an issue with the engine itself, and not actually anything being experimented with. Additionally, as a software developer, developing with blueprints was simply very slow. The UI for triggering simple functions or adjusting simple variables took several interactions, which could be

replaced with a simple single line of source.

After multiple weeks of trying to use the slow Blueprints development tool, the research focus shifted towards figuring out how to manipulate anything we can manipulate with Blueprints, with the C++ source instead. The key issue here was making sure any source added did not interfere with the normal operation of the rest of the view that was generated. Previous experience with game engines foretold that there would be several events the game handles at run time, and they could be overwritten to run the commands needed for the project. After using the same sources, the Tick event was the event that fulfilled the teams needs. The Tick event fires at every frame, and would safely execute the source. From here, all research was devoted to converting Blueprint concepts that were planned out into source.

In terms of addressing the specific issues that were described, there were solutions available, although not straightforward. Naturally, there were other programmers that were developing with the Unreal Engine, and they had their ways of manipulating the environment with source instead of Blueprints. There were several solutions, but nothing was standard. Each solution had its disadvantages, and much thought and time went into making sure the chosen solution did not negatively impact our project goals.

Works Cited

"UE4 AnswerHub - UE4 AnswerHub." UE4 AnswerHub - UE4 AnswerHub. N.p., n.d. Web. 07 May 2015.

"Unreal Engine Community Wiki." Epic Wiki. N.p., n.d. Web. 07 May 2015.

"Unreal Engine." YouTube. YouTube, n.d. Web. 07 May 2015.

Solution (Group/Individual)

The final, detailed solution is to overwrite the Tick event of the first-person perspective that belonged to the user. Since there would be only one user on each end, since the Unreal Engine is currently designed to only handle one Oculus Rift unit at a time. Therefore, all decision for what objects that the user can see, and what models should be rendered where from the user's perspective will be calculated frame by frame in this event. The alternative was to create an object in the environment that would handle where every object is in the world, but creating this object proved to be too difficult. Every attempt at creating this object resulted in the Unreal Engine refusing to compile. Every other solution researched involved having some world object that managed where objects in the room were every frame, but the solutions could not be implemented. The solution implemented is simply cutting out the middleman, so to speak, which was ultimately an advantage, despite being less modular.

Some problems that arose from our solution include the inability to change certain settings by source code. Certain physics properties, and all key input bindings hand to be change from within the engine's UI. Any attempt to change them in the source was overwritten by the Unreal Engine UI's default properties, and they were not easily reversible without crude fixes that would be better to avoid. Our inability to control when Unreal engine adjusts properties exposed another issue: scalability.

Every frame, data needs to be read and checked. If any object is created or moved, it is then processed. If there are thousands of objects in the virtual reality room, and each of them was moving every frame, thousands of objects need to be processed. There are two solutions to this: only manipulate objects within reasonable proximity of the user, or multi-threading. The proximity solution does not always solve the issue, and it was not implemented at all in fear that proximity calculation may actually be doing more work, since it is out of the programmer control how much work is actually being done when issuing commands to the Unreal Engine. The multi-thread solution would work, however mutual exclusion becomes a serious concern since, once again, there is no control over when the Unreal Engine is manipulating certain data.

Mutual exclusion was a concern for the project as a whole as well. Plenty of tools were implemented to make sure any data sent by VIVE is not lost or altered by the Unreal Engine. The project as a whole is very modular, using the Model-View-Controller design pattern. The Unreal Engine source plays the role of the view, but it implements no design pattern internally due to the limited control the programmer actually has. It is, however, developed to be as modular as possible internally. It is very well-structured and organized, despite the aforementioned sacrifice made when not creating a world object. This flexible design has proved to be very efficient and it has been stress-tested successfully. Despite a large amount of data to process into the view, there seems to be minimal latency. Even though scalablilty was a concern initially with our solution, it is quickly becoming less and less of a concern. The one remaining concern for the project is manipulating settings from within the Unreal

Engine UI, since it may overwrite settings the source tries to set. These changes can easily affect and break the project due to the chaotic effects of changing simple solutions. The Unreal Engine does not have strong decoupling. These concerns are sincere and need addressing since the project is a framework that can ideally be expanded to implement all sorts of virtual reality projects. The plan is to make the source available to expand.

Source Code (Group/Individual)

The source is located at a group of repositories found here: https://github.com/NCSALOVR. The repository labeled Unreal contains all the source for the view: https://github.com/NCSALOVR/Unreal. Unfortunately, it is not simple to make use of the source straight from the repository. There are two core things that need to be done to make use of the source. First, the Unreal Engine must be the correct version. Unreal projects have little portability, and so version 4.4.3 must be used. Second, once launching the correct version of the Unreal Engine, create a new project of type "Code First Person." This will launch Visual Studios C++ 2013. from there, we can adjust the source of each file, and all source in the Public and Private directories need to be replaced with the source from the repositories.

For future version control, the Unreal UI has options for version control that can be looked into. To start expanding on the project, observe ViveViewCharacter.cpp, and notice the Tick function. This is what can be manipulated to make adjustments to what happens every frame. You will also need to eliminate all keybindings within the Unreal Engine UI. To bring the project from source to UI, right click the project and

select the debug option, to run the project. If everything compiles, the UI will launch and you will be able to edit the environment from the editor. It is also recommended that you delete all visible objects in the room, so that the provided source is doing all of the rendering.

From here, the code is ready to be expanded on. If editing from the Unreal Engine is desired, it should not interfere with the source usually, since the final design was very modular. However it is risky, and changing some settings or properties with the UI or Blueprints has broken the project on certain occasions. Only proceed to do so if there is a good understanding of the Unreal Engine. It is also smart to understand how the VIVE and the server source works. Please take a look at their repositories to make sure there is a clear understanding of the role of each piece of source.

Once everything is set up, launch the server code, launch the appropriate software from VIVE, and then launch the ViveView from the Unreal UI by clicking Launch. If there are issues of getting the data between all of the parts, make sure the file paths in the source code match between all parts of the project. The Unreal engine currently writes and reads at a very specific location, and it is crucial that VIVE writes and reads the same location. Feel free to adjust these locations to your needs. For the Unreal Engine part of the project, all file paths are in the UpdateFromVive function. For more information on what needs to be adjusted for the other parts of the project, read the descriptions for other individual group parts. Because of the strong division of labor, the team is fairly modular.

Ankoor Shah
Final Report – cs493
amshah4

Your Team Experience (Group/Individual)

The division of labor was done early on based on the specialties of the team members. While not completely detailed, the long term goals of the project were clear early on, and it was decided there would be a team for the model, the view, and the controller. With 6 members, three teams of two were created, with three separate repositories. Due to the early planning, as long as each team's portion successfully worked, the project as a whole would work with little difficulty. After planning, the team split off into their own groups. Weekly meetings kept everyone in touch on progress, and daily scrum-like informal conversations kept everyone on the same page in terms of daily progress.

Within teams, issue tracking and handling was all done through informal conversation, as aforementioned, unless the issue became a large problem that took multiple days to properly address. In such a case, the Github issue tracker was used to communicate problems. However, keeping the issue tracker up to date was not consistently done, and it mostly lead to busy work as opposed to a helpful organization tool. Simply conversing with each other and tossing ideas in the air helped address the issue more than anything else.

During these conversations there were no clear leaders. No one had a final say or a clear choice in direction. Each team respected each other's roles and difficulties and offered advise when possible. During each of these conversations the team would be reminded about the amount of time left, and the tasks that still need to be done, and that seemed sufficient to motivate the team to remain productive. This worked for most of the project. Once the project hit the phase where all parts needed to be

assimilated, the modular team began to blur all the roles, since an understanding of all parts was necessary to run and test the project as a whole, but the productivity was not negatively impacted. Simply communicating kept everyone willing to contribute.

Communication with your Client (Group/Individual)

Communication was very key to our group, and that included communication with the liaison, who essentially took the role of our client. Communication was scheduled weekly, and were basically a summary of our aforementioned daily conversations. Our liaison did a good job of sincerely reminding everyone of the remaining time and goals, and potential ideas and extra features that would significantly enhance the project and productivity if it did not impact the workload significantly. This kept the team constantly thinking and open-minded, while continually having us analyze how much time remained for all remaining work. There was a clear schedule for the project that was developed in the first few meetings with the client, and therefore every meeting was pretty clear on what to discuss. Any visual resources that were used to explain anything in the meeting were added to the wiki, so that they can be viewed by anyone at any time.

Lessons Learned as a Team (Group/Individual)

Sharing resources that can help explain anything is very crucial. As a team, we really learned that it is difficult to provide advice that isn't incredibly general or unspecific unless there is very clear understanding of the problem. When any part of the project

was not working, calmly figuring out who to contact and what to ask was crucial to narrowing down the problem and its location. This speeds up productivity significantly, something the team definitely could have used. There were several small improvements that the team wanted to implement, but we simply did not have the time to do. It would be excessive to say the team's productivity was overestimated, since features like dynamic asset selection and multiple rooms were considered extra features since the beginning of the project. Overall the team did dodge most traps and pitfalls of working on a project together, and we maintained productivity.

Lessons Learned as an Individual (Individual)

As an individual in the team, there were two main issues faced. The demeanor in which the team was addressed may have been occasionally too judgmental. This was not done with ill-intention. Whenever there was an issue to address, the person in charge of the most relevant source would understand the problem the best. Due to poor phrasing, it is possible this person was referred to as though they were being blamed. Communication skills were practiced to improve communication with the team. The other issue had to do with self-confidence. Performance in the team was exactly what would be expected and planned, however the feeling never left that more could be done. More features to add, more communication could be done, or anything other work completed can be further improved. The aforementioned communication issue was never really acknowledged by the team more than myself, and it may not have truly been an issue. Being aware of what is actually negatively or

positively impacting the team is the primary skill that needs improvement. As these skills improved, the ability to work well in the team improved and it helped bring the project to completion.

Recommendations for Improvement or Extension (Individual)

That summarizes the entire project experience! It was very engaging and exciting, and our project is now ready to be expanded on by anyone who is ready and willing. During the development of this project the Unreal Engine became free of charge, and so all tools needed for the project are publicly available. One of the core fundamental ideas of the project is to keep it like a framework so it is easy to expand on and continue to develop. For questions on making use of the source, or clarifications on anything not thoroughly explained, please contact me at ankoor.m.shah@gmail.com.