

# Orbital Simulation Suite

## Final Report

Cougs in Space



### **Mentor:**

Aaron Crandall & Cougs in Space

### **Team Phoebe**

Carly Ott, Elijah Craig, Jesslyn Khamsoi, Nathaniel Nemiroff

CptS 423 Software Design Project II  
Spring 2018

Instructor: Aaron Crandall

<b>I. Introduction</b>	<b>3</b>
<b>II. Team Members &amp; Bios</b>	<b>4</b>
<b>III. Project Requirements Specification</b>	<b>5</b>
III.1. Project Stakeholders	5
III.2. Use Cases	5
III.3. Functional Requirements	5
III.4. Non-Functional Requirements	5
<b>IV. Software Design - From Solution Approach</b>	<b>7</b>
IV.1. Architecture Design	7
IV.1.1. Overview	7
IV.1.2. Subsystem Decomposition	8
1. STK Application	8
2. Docker Image with Google Cloud Build	8
3. Slackbot	8
IV.2. Data design	8
IV.3. User Interface Design	9
<b>V. Test Case Specifications and Results</b>	<b>10</b>
V.1. Testing Overview	10
V.2. Environment Requirements	10
V.3. Test Results	10
<b>VI. Projects and Tools used</b>	<b>11</b>
<b>VII. Description of Final Prototype</b>	<b>12</b>
VII.1. Docker and CI (Component 1)	12
VII.1.1. Functions and Interfaces Implemented	12
VII.1.2. Preliminary Tests	12
VII.2 STK Application	13
VII.2.1. Functions and Interfaces Implemented	13
VII.2.2. Future work	13
VII.3. Slackbot	14
VII.3.1 Functionality	14
<b>VIII. Product Delivery Status</b>	<b>14</b>
<b>IX. Conclusions and Future Work</b>	<b>16</b>
IX.1. Limitations and Recommendations	16
IX.2. Future Work	17

<b>X. Acknowledgements</b>	<b>17</b>
<b>XI. Glossary</b>	<b>18</b>
<b>XII. References</b>	<b>18</b>
<b>XIII. Appendix A – Team Information</b>	<b>19</b>
<b>XV. Appendix B - Project Management</b>	<b>19</b>
<b>XVI. Appendix C - Docker/CI Images</b>	<b>21</b>
<b>XVII. Appendix D - Instructions for future workers</b>	<b>23</b>

## **I. Introduction**

Washington State University's cube satellite club Cougs in Space has requested our team work on a simulation suite for their future cube satellites. Our mentor, Dr. Crandall, requests that this be done in the Systems Tool Kit, a state of the art tool used widely in industry by some of the most well known companies such as Boeing and Lockheed Martin. The orbital simulation suite, has a goal of being able to take current designs developed by CIS and run a simulation of that design in orbit.

This OSS is then to be integrated into CIS's GitHub remote repository continuous integration/continuous deployment tool. When new or updated designs are accepted into the repository, the CI/CD must run this simulation and deliver data logs containing mission relevant information. These data logs will contain important information about the CubeSat's solar panels obscurations, predicted power generation, solar intensity, and affected area. These numerous reports will notify cougs in space through a customized slack bot which will post these reports to the club's thread. The ability to extensively test their cube satellites designs will ensure that CIS launches well vetted and operational equipment.

Our mentor and the leader of Cougs in Space, Aaron Crandall is available for contact through [acrandal@wsu.edu](mailto:acrandal@wsu.edu).

## **II. Team Members & Bios**

Elijah Craig is a computer science student interested in operating systems, embedded systems, and networking. He works as team lead for Team Phoebe, and is lead developer of model files and a codeveloper of implementing the STK components. He is also the CIS team advisor for computing systems. His skills are C/C++, C#, and RTOS programming.

Carly Ott is a computer science student interested in game development and data analysis. On this project, she is the developer for the deliverable component; creating a functional Docker image that runs STK and implementing continuous integration. She has experience with C/C++, C#, and python programming.

Jesslyn Khamsoi is a computer science student interested in software engineering for websites and gaming. She is a co-developer of implementing the STK components, consisting of inserting a satellite object, running an orbital simulation, and generating reports. She has experience programming in C/C++, C#, and Python.

Nathan Nemiroff is a computer science student with a desire to work on software development and server maintenance. In this team he developed a Docker image that allows systems like Dockerhub and Github to utilize it to push update notifications to a slack channel by taking advantage of the webhooks features. He's worked frequently in C++, C#, and Python in the past.

# III. Project Requirements Specification

## III.1. Project Stakeholders

The primary stakeholder for this tool, Dr. Aaron Crandall who started the cubesat project, and the Cougs in Space club to assist in a successful satellite launch and give WSU it's first satellite. Cougs in Space will repeatedly submit a updated design of the cubesat to an OSS testable branch. Once the design reaches the GitHub server, the continuous integration tool will activate our OSS, and will run through all changed files, and all files dependent on those files, and will build a virtual representation of the updated cubesat. Once built, it'll run it through extensive testing, simulating the orbit, and the satellite's solar panel solar exposure that will affect the satellite's generated power. It'll then return a data log of the efficacy of the new design. Based on the reports, Cougs in Space will gain information on how to improve their model if needed.

## III.2. Use Cases

The primary use case is to function as an addition to the current CIS structure team workflow. The process starts after a successful design review has been completed for the team and the new revision is uploaded to the repository. Upon update, the CI functionality will take place where the new design is converted to a COLLADA file that acts as the digital representation in STK. The simulations occur and reports are generated containing data such as heat fluxuations, sun exposure, and access to ground stations. These are then delivered to the "STK\_Sim" channel within CIS's Slack workspace where the structure team can review and infer additional data, as well as seeing if the simulation detected critical failures.

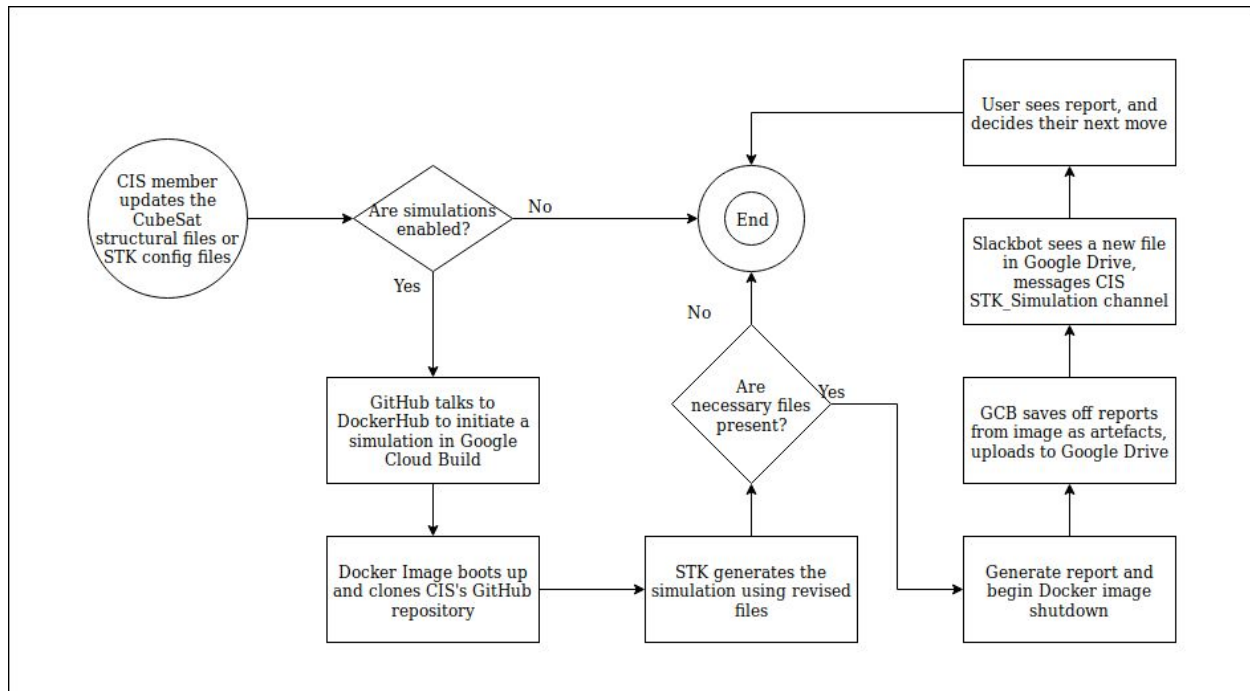
## III.3. Functional Requirements

- OSS must convert from whatever filetype CIS Structure team works in to a STK compatible filetype (.dae, .mdl, .gitf)
- OSS must be able to simulate orbital data such as access, sun obscuration, and report solar climate (for structure team to infer from)
- OSS must be able to calculate solar intensity and sun exposure
- OSS must exist on a virtual machine that can be booted upon every new revision
- OSS must use the most recent iteration of structural design
- OSS must take as input a config file containing pertinent satellite data (mass, material)
- OSS must save off reports to a cloud storage service
- OSS must update the Slack workspace upon completion

### **III.4. Non-Functional Requirements**

- OSS should be extendable to allow for new tests and simulations
- OSS's virtual image should be modifiable by remote connection
- OSS should store many iterations of reports
- OSS should report failures of the simulation at runtime and compile time

## IV. Software Design - From Solution Approach



### IV.1. Architecture Design

Revise and include Section II from your Solution Approach report here. Provide the block diagram of your architecture and give a brief description of it.

#### IV.1.1. Overview

The attributes of the satellite test suite has the ability of testing systems/C&DH, structures, thermal, power and attitude, but currently only tests power generation. In order to test power we will be using the the STK Gfx Analysis Solar Panel tool, which was supplied in STK's code samples. This tool is able to predict the amount of power taken in from the solar panel by the satellite's simulation for an amount of given time. The solar panel's sun exposure for the amount of time is calculated which then estimates the amount of electrical power(w) the satellite will have for use. The tool has both a solar panel view and solar panel window. The solar panel view window uses animation to show the change in the panel's position, while the other window takes in the time period of the model, identifies unknown objects that block the panel's exposure, and displays the power results. The results consist of area (area and intensity of panels used), angles (angle and exposed areas of the panels), and power (units of obtained power and intensity of sun rays), which can be displayed in report or graph form showing these attributes at

different times over the time period. The results are currently displayed in report form, which is generated from the java files containing the tool's commands. For our final deliverable we use the AGI STK analysis tools to test for energy production based on obscuration between the sun and the solar panels, the total activated area, and predicted generated wattage of the solar panels. related failures upon simulated launch. The tests will be implemented in Java, as we have found this language is able to handle the implementation requirements better than C# which was our previous attempted implementation language. The test suite will be contained in the CiS public Github repository once the team retrieves the project files from the github accounts used by team Phoebe. The tests will automatically run whenever a model change is pushed to the repo. This will ensure that all commits to the CiS repository are tested, allowing the user to know if their changes have made the satellite simulation viable. We plan on exporting the data to users via Slack workspaces and channels. CiS decided to have us develop their test suite instead of using a pre-existing software because of customization. Our deliverable has been built upon the tool kit used by the U.S. Air Force to simulate satellites in their Joint Space Operations Center. The software is able to be reverse compatible with all of the saved previous CiS models and data to provide maximum functionality.

## **IV.1.2. Subsystem Decomposition**

### **1. STK Application**

This subsystem is in charge of the actual simulations. It is capable of taking as input the CIS repository and constructing the STK compatible CubeSat then delivers the reports of the simulations.

### **2. Docker Image with Google Cloud Build**

This is a remotely hosted virtual machine that contains STK, our STK Application, and will be the location where all tests are performed. Upon booting it fetches the newest CIS repository which it will deliver to the STK Application. After the reports are generated by STK, the VM will save off the reports as artifacts which will be located in Google Cloud Build's storage. This allows a report to be directly accessed by build number, and will keep several iterations of reports.

### **3. Slackbot**

The slackbot is called upon when GitHub detects the CI function is terminating. It will send an update to the CIS Slack Workspace in the dedicated channel with either a link to the build, or just the build number.

## **IV.2. Data design**

Describe all data structures (including the internal and temporary data structures), and the database(s) created as part of the application.



In the STK application responsible for generating reports, a “Data” class used to compute solar panel analysis outputs require the following variables: Satellite’s title, report type, start and Stop time of simulation, satellite's dimensions, and time step. A “Compute” function takes in an instantiated data class and calculates the solar intensity, affected/activated area ( $m^2$ ), and power(w).

STK runs a simulation in a tree structure manner, a root Class is instantiated, which has the functionality to run a Scenario. The scenario acts a node with child nodes of any object with in the orbit simulation (in this case the cubesat). A scenario is ran from the Start time to the Stop time defines in the “Data” class. Based on the exposure between the sun and the solar panels, on the satellite model (dimensions defined in “Data” class) analysis data (solar intensity, affected/activated area ( $m^2$ ), and power(w)) are calculated.

### **IV.3.User Interface Design**

There is no direct GUI.

# **V. Test Case Specifications and Results**

## **V.1. Testing Overview**

Testing is a bit enigmatic for the OSS. Many parts are either tested sufficiently by AGI, too complex to test, or it is just output to input piping. What is certain that what is feasibly testable will be predominantly unit tests. A scenario within STK may be generated by a test's config file, so testing the creation of sample data and then scanning said scenario to ensure all objects are preset would be the manner in which to test. Using the tested config file would then be used in the report generator. This would determine if the config file will invoke errors when running the computations.

## **V.2. Environment Requirements**

Specify both the necessary and desired properties of the test environment. The specification should contain the physical characteristics of the facilities, including the hardware, the communications and system software, the mode of usage (for example, stand-alone), and any other software or supplies needed to support the test. Identify special test tools needed.

Our testing environment should be in the same environment as the OSS, as many of the testable portions such as piping are dependent on that image. This will require a Ubuntu 18.04 image containing openjdk-8-jdk as it's default Java as well as containing STK11.5.0 and our source code. This can be sufficiently tested on a Raspberry Pi 2, so the actual hardware requirements are minimal.

## **V.3. Test Results**

Our tests proved to be very minimal, for both time and minimal need reasons. Our main results were found by manual testing and attempting to break the app. We found that OSS is currently incapable of taking COLLADA file types for our simulations, which was an issue we did not have time to fix. We also found that as scenarios are generated they are perfectly represented, which came as no surprise. This was a very high level unit test.

## VI. Projects and Tools used

Tool/library/framework	Quick note on what it was for
Apache Ant	The build and launch platform
Github	The source code hosting platform
DockerHub	Contains the Docker image that has STK engine/libraries
Docker	Used to build/run the STK engine with environment variables set
Google Cloud Storage	STK build artifacts are saved here (on Google Cloud Platform)
Google Cloud Build	The host used in order to run/build our Docker image
STK by AGI	The simulation software
Python	The language used to write the SlackBot
Java	The language used to write the STK App

Languages Used in Project		
Java	Python	C# (Later changed to Java)

# VII. Description of Final Prototype

## VII.1. Docker and CI (Component 1)

### VII.1.1. Functions and Interfaces Implemented

Our Docker image is essentially a barebones STK Virtual Machine that can run on almost any computer. It is currently hosted on a public Docker repository, able to be pulled with only a file size of 5 GB. The image that contains the most updated code is carlyishere/stk\_cis:final. In order to have such a small overall size comes at the cost of functionality, there is no GUI of any kind and exclusively a command line interface. There was no text editor of any kind on the base image, and many common linux commands would have to be manually installed. Our Docker image has many STK Java code samples that can be run without the graphical interface. Each code sample is located in a different location or file within our Docker image. Depending on which code that needs to be tested, the Dockerfile (located in our GitHub repository) will need to be updated. The location of the code to be tested and the command to run the code will need to be known. Everything else that is needed to test the different code samples are either downloaded within our Docker image, or set within our Dockerfile.

### VII.1.2. Preliminary Tests

```
Buildfile: /home/OrbitalSimulationSuite/NoGraphics/build.xml
Total time: 12 seconds
BUILD SUCCESSFUL

run:

[java] done
[java] 16 1 May 2019 18:41:44.740628652 1 May 2019 19:00:00.000000000 1095.2593713483366 16 N/A 6.8581890134292225 -132.8354571006107 500.3055639525988 44.15571659114222 -69.48692002757399 510.3380935783529 -40.053724265214484 -19.671219643485795
9.9999999999993648E16 -40.053724314543004 -24.247294586228648 9.9999999999993616E16 CougSat - LineOfSight Object Availability Time 16 N/A To Star Sun From Satellite CougSat No Parent No Parent 16.026425 N/A 16.219355 N/A 1 N/A 1.000000 N/A 1.000000 N/A N/A
N/A 16.122890 N/A 08N 739.203808988128 758.6119480898684 08NQ3920358612 197 461.06385099009026 4889.2834259808787 19TDJ6106489283 27G 613.3371199252648 5565.434085588036 27GKR1333765434 26G 734.8015580466918 5562.648737905443 26GQA3480262649
[java] 15 1 May 2019 17:07:07.795299122 1 May 2019 18:36:00.495246183 5332.699947060522 15 N/A 6.8598198310284975 -109.11510197366243 500.3057071916847 -8.623702958898674 -146.93099290253062 500.48009756737355 -40.05372401734073 4.047479327016718
9.9999999999993648E16 -40.05372424977127 -18.23293689172964 9.9999999999993648E16 CougSat - LineOfSight CougSat - LineOfSight 15 N/A To Star Sun From Satellite CougSat No Parent No Parent 15.026431 N/A 15.965786 N/A 1 N/A 1.000000 N/A 1.000000 N/A N/A
15.496109 N/A 12N 708.2871798461623 758.6618046769863 12NYN0828758662 06L 507.5925226365292 9046.749070902217 06LMR0759346749 31G 589.3432755207214 5565.754339536023 31GER8934365754 27G 736.0264124184235 5562.610748666124 27GVR3602626211
[java] 14 1 May 2019 15:32:30.844981474 1 May 2019 17:01:23.492119249 5332.647137775333 14 N/A 6.861227232011066 -85.39495286485531 500.30583083435556 -8.624656633349039 -123.2130330727771 500.48020231566517 -40.053723785780655 27.766199153970362
9.999999999999368E16 -40.053724002835544 5.486003570274068 9.9999999999993632E16 CougSat - LineOfSight CougSat - LineOfSight 14 N/A To Star Sun From Satellite CougSat No Parent No Parent 14.026437 N/A 14.965783 N/A 1 N/A 1.000000 N/A 1.000000 N/A N/A
14.496110 N/A 16N 677.3536488172726 758.7048881411218 16NFN7735458705 10L 476.560996800506986 9046.637788487344 10LDR7656146638 35G 565.3515845184992 5565.998741167156 35GNR6535265999 31G 712.04968915771 5563.318575143048 31GGR1205063319
[java] 13 1 May 2019 13:57:53.890848967 1 May 2019 15:26:46.483534714 5332.592685747004 13 N/A 6.862463765822542 -61.67496132028329 500.3059394865819 -8.625853666106348 -99.49530073838584 500.4803338094986 -40.05372357018729 51.48493490446808
9.9999999999993632E16 -40.05372377225149 29.20496685042478 9.9999999999993616E16 CougSat - LineOfSight CougSat - LineOfSight 13 N/A To Star Sun From Satellite CougSat No Parent No Parent 13.026442 N/A 13.965778 N/A 1 N/A 1.000000 N/A 1.000000 N/A N/A
13.496110 N/A 20N 646.4070791937531 758.7470649438872 20PNK4640758747 44L 445.5048844520325 9046.476656451292 14LMR4550446477 39G 541.3615256081507 5566.167324615143 39GWR4136266167 35G 688.075964762446 5563.950497873307 35GPR8807663950
[java] 12 1 May 2019 12:23:16.943587889 1 May 2019 13:52:09.483264955 5332.539677066416 12 N/A 6.86400804873855 -37.954685941939346 500.3060752072044 -8.626679924830245 -75.77722183051468 500.4804245842007 -40.05372370205766 75.20364196235371
9.999999999999368E16 -40.053723557613644 52.923895308927455 9.9999999999993648E16 CougSat - LineOfSight CougSat - LineOfSight 12 N/A To Star Sun From Satellite CougSat No Parent No Parent 12.026448 N/A 12.965775 N/A 1 N/A 1.000000 N/A 1.000000 N/A N/A
N/A 12.496111 N/A 24N 615.4953799680868 758.8413948914625 24NXN1549558841 18L 414.4841474483748 9046.33363423491 18LVR1448446334 43G 517.3691927866939 5566.260121800104 43GER1736966260 39G 664.0997244004645 5564.506680075412 39GXR6410064507
[java] 11 1 May 2019 10:48:39.993379095 1 May 2019 12:17:32.474389721 5332.481010626208 11 N/A 6.8654208301044256 -14.234532360621238 500.3061993504368 -8.627889910147285 -52.059501630714024 500.4805575312221 -40.05372318547245 98.92236135600585
9.9999999999993648E16 -40.05372335856556 76.64285984986901 9.9999999999993648E16 CougSat - LineOfSight CougSat - LineOfSight 11 N/A To Star Sun From Satellite CougSat No Parent No Parent 11.026454 N/A 11.965770 N/A 1 N/A 1.000000 N/A 1.000000 N/A N/A
11.496112 N/A 28N 584.5731446088772 758.939124210266 28NEN8457358939 22L 383.4229992296098 9046.125157448929 22LCR8342346125 47G 493.377848671676 5566.27711829001 47GMR9337866277 43G 640.1274033318837 5564.987016337404 43GFR4012764987
[java] 10 1 May 2019 09:14:03.041189933 1 May 2019 10:42:55.473789681 5332.432599747532 10 N/A 6.866743848627695 9.485539539067156 500.3063157188782 -8.628730889517868 -28.341436505844616 500.4806499444023 -40.053723015616434 122.64108903024739
9.9999999999993648E16 -40.05372317474407 100.36178977622315 9.9999999999993648E16 CougSat - LineOfSight CougSat - LineOfSight 10 N/A To Star Sun From Satellite CougSat No Parent No Parent 10.026459 N/A 10.965766 N/A 1 N/A 1.000000 N/A 1.000000 N/A N/A
N/A 10.496113 N/A 32N 553.644038310174 759.0450779024601 32NNN5364459045 26L 352.39714445331236 9045.934622973626 26LLR5239745935 51G 469.3874553143235 5566.218323741661 51GVR6938766218 47G 616.152721522619 5565.391657480702 47GPR1615365392
```

We made our Docker image easier to run by implementing Google Cloud Build CI. Our Docker image is located in DockerHub, our Dockerfile CloudBuild file is located in the GitHub CarlyOtt/CIS\_Docker\_Repo repository, and Google Cloud Platform connects them all in order to make this process a lot easier. Upon pushing or committing to the GitHub repository, it will trigger an automatic build set on Google Cloud Build. Google Cloud will go to the specified

GitHub repository and find the CloudBuild.yaml file. From there, it will trigger the Dockerfile and run the Docker image that is located on the DockerHub account. During the build, it will save the created artifacts and store them in a file in the Cloud Storage. Unfortunately, the reports created during these runs were unable to be saved during these builds. However, the reports are printed out in the build logs and these build logs can be located and downloaded. This can be seen in the picture provided above. Right now, a complete and successful build takes about four minutes to complete.

## VII.2 STK Application

### VII.2.1. Functions and Interfaces Implemented

The STK application ended up being distinct tests that can be individually executed. All tests are heavily reliant on STK's code samples, as any useful documentation is hard to come by online. That coupled with the heavy graphical integration and the general application nature of STK means the overall processes are more memory and CPU intensive than we would prefer. However the functionality is entirely there, such that a simulation is successfully ran, and data logs are saved (pictures of reports below). The manner in which to use the simulations involves locating the proper model file (not COLLADA) within the config file and then running the project with ant. All tests currently only depend on that model file's location, and nothing else. All cities, satellite constellations, etc. are hardcoded in as those requirements are not likely to change and would involve a larger process to be undertaken by the structures team. There currently is no attitude support, though that was decided to be a more reliable feature towards the end of the project.

Time (UTC)	Effective Area (m <sup>2</sup> )	Solar Intensity
Apr 2019 19:00:00.000	27.853827	1.00000030
Apr 2019 19:05:00.000	27.851852	1.00000030
Apr 2019 19:10:00.000	27.851852	1.00000030
Apr 2019 19:15:00.000	27.849877	1.00000030
Apr 2019 19:20:00.000	27.849877	1.00000030
Apr 2019 19:25:00.000	27.847901	1.00000030
Apr 2019 19:30:00.000	27.849877	1.00000030
Apr 2019 19:35:00.000	27.851852	1.00000030
Apr 2019 19:40:00.000	0.000000	0.00000030
Apr 2019 19:45:00.000	0.000000	0.00000030
Apr 2019 19:50:00.000	0.000000	0.00000030
Apr 2019 19:55:00.000	0.000000	0.00000030
Apr 2019 20:00:00.000	0.000000	0.00000030
Apr 2019 20:05:00.000	0.000000	0.00000030
Apr 2019 20:10:00.000	0.000000	0.00000030
Apr 2019 20:15:00.000	27.851852	1.00000030
Apr 2019 20:20:00.000	27.849877	1.00000030
Apr 2019 20:25:00.000	27.849877	1.00000030
Apr 2019 20:30:00.000	27.847901	1.00000030
Apr 2019 20:35:00.000	27.847901	1.00000030
Apr 2019 20:40:00.000	27.845926	1.00000030
Apr 2019 20:45:00.000	27.847901	1.00000030
Apr 2019 20:50:00.000	27.847901	1.00000030
Apr 2019 20:55:00.000	27.849877	1.00000030
Apr 2019 21:00:00.000	27.845926	1.00000030
Apr 2019 21:05:00.000	27.847901	1.00000030
Apr 2019 21:10:00.000	27.849877	1.00000030
Apr 2019 21:15:00.000	0.000000	0.00000030
Apr 2019 21:20:00.000	0.000000	0.00000030
Apr 2019 21:25:00.000	0.000000	0.00000030
Apr 2019 21:30:00.000	0.000000	0.00000030
Apr 2019 21:35:00.000	0.000000	0.00000030
Apr 2019 21:40:00.000	0.000000	0.00000030

Time (UTC)	Power (W)	Solar Intensity
19:00:00.000	5323.170	1.00000030 Apr 2019
19:05:00.000	5322.793	1.00000030 Apr 2019
19:10:00.000	5322.793	1.00000030 Apr 2019
19:15:00.000	5322.415	1.00000030 Apr 2019
19:20:00.000	5322.415	1.00000030 Apr 2019
19:25:00.000	5322.038	1.00000030 Apr 2019
19:30:00.000	5322.415	1.00000030 Apr 2019
19:35:00.000	5322.793	1.00000030 Apr 2019
19:40:00.000	0.000	0.00000030 Apr 2019
19:45:00.000	0.000	0.00000030 Apr 2019
19:50:00.000	0.000	0.00000030 Apr 2019
19:55:00.000	0.000	0.00000030 Apr 2019
20:00:00.000	0.000	0.00000030 Apr 2019
20:05:00.000	0.000	0.00000030 Apr 2019
20:10:00.000	0.000	0.00000030 Apr 2019
20:15:00.000	5322.793	1.00000030 Apr 2019
20:20:00.000	5322.415	1.00000030 Apr 2019
20:25:00.000	5322.415	1.00000030 Apr 2019
20:30:00.000	5322.038	1.00000030 Apr 2019
20:35:00.000	5322.038	1.00000030 Apr 2019
20:40:00.000	5321.660	1.00000030 Apr 2019
20:45:00.000	5322.038	1.00000030 Apr 2019
20:50:00.000	5322.038	1.00000030 Apr 2019
20:55:00.000	5322.415	1.00000030 Apr 2019
21:00:00.000	5321.660	1.00000030 Apr 2019
21:05:00.000	5322.038	1.00000030 Apr 2019
21:10:00.000	5322.415	1.00000030 Apr 2019
21:15:00.000	0.000	0.00000030 Apr 2019
21:20:00.000	0.000	0.00000030 Apr 2019
21:25:00.000	0.000	0.00000030 Apr 2019
21:30:00.000	0.000	0.00000030 Apr 2019
21:35:00.000	0.000	0.00000030 Apr 2019
21:40:00.000	0.000	0.00000030 Apr 2019

### VII.2.2. Future work

As the project concluded it was determined that STK isn't the only tool necessary to complete the ideal function of the OSS. The OSS will need a true physical simulation software such as Blender or Solidworks. With the education license of STK, we simply do not have access to the

higher simulation functions STK can provide. There is also the potential issue of those not being programmatically available, as we found no reliable API to suggest this. This aligns with our understanding of STK being an “application first” type of software.

In tandem with Blender, we speculate that the positional data provided by STK could allow for highly accurate heat transfer and solar intensity tests.

## VII.3. Slackbot

### VII.3.1 Functionality

With no report to fetch, the slackbot encountered some conceptual issues. What is capable is a message to a Slack workspace that can be tied to GitHub as the CI terminates. This is sufficient to alert CIS that the revision has been tested and the reports are ready to be accessed. A number of methods to capture the data to be parsed and reported to the slack can be implemented depending on the desired CI path and what technologies are used after test is run on Docker.

## VIII. Product Delivery Status

All source code, documentation, and reports will be uploaded to a CIS hosted repository by May 2nd, 2019.

- Repositories needed in order to run the STK tests
  - GitHub:
    - Repo: ElijahBryonCraig/OrbitalSimulationSuite
    - Branch: Master
- Repositories needed in order to run the Docker image:
  - GitHub:
    - Repo: CarlyOtt/CIS\_Docker\_Repo
    - Contains dockerfile and cloudbuild.yaml file
    - [https://github.com/CarlyOtt/CIS\\_Docker\\_Repo](https://github.com/CarlyOtt/CIS_Docker_Repo)
  - DockerHub:
    - Contains Docker image with STK engine
    - Account: carlyishere
    - Image and tag: carlyishere/stk\_cis:final
    - [https://cloud.docker.com/u/carlyishere/repository/docker/carlyishere/stk\\_cis](https://cloud.docker.com/u/carlyishere/repository/docker/carlyishere/stk_cis)
    - Builds are connected to the CIS\_Docker\_Repo repository.
  - Google Cloud Platform:
    - Project name: DockerCIstk

- Project ID: cpts423seniorproject
  - Artifacts and Images and Files are saved in Cloud Storage (under Storage)
  - Builds and build logs can be seen in Cloud Storage then History (under Tools)
  - Triggers are set in Cloud Build then Triggers (under Tools)
  - <https://console.cloud.google.com/home/dashboard?project=cpts423seniorproject>
  - Connects with the CIS\_Docker\_Repo repository.
- How to run the STK Docker image:
  - Make sure you have access to a DockerHub account that contains the carlyishere/stk\_cis:final image.
    - Create a new account and pull the image from our DockerHub account. Save the image to the new account.
  - Have a GitHub repo that contains the dockerfile and the cloudbuild.yaml files.
    - Can just copy or pull those files from the CarlyOtt/CIS\_Docker\_Repo repo and put them somewhere in your GitHub account.
      - Make sure that the dockerfile is named "Dockerfile" exactly (and not Dockerfile.txt. etc). Also, make sure the CloudBuild.yaml file is spelled exactly the same as ours (CloudBuild.yaml).
  - Make sure both the DockerHub and GitHub accounts are connected.
    - In DockerHub, go to "Builds", and click the blue button "Configure Automated Builds". Connect your GitHub account with this.
    - Next in DockerHub, go to "Webhooks" and create a webhook for your GitHub account. Here's how ours looked:  
[https://github.com/CarlyOtt/CIS\\_Docker\\_Repo.git](https://github.com/CarlyOtt/CIS_Docker_Repo.git).
    - You can force a trigger in DockerHub under "Builds" in order to make sure both accounts are set up together properly by pressing the blue and white "Trigger" button.
  - Important: in order to use Google Cloud Platform/Build, some payment method needs to be entered into the account (like club funding). The account gets \$300 worth of free credit for a year (over 300 days left on that free trial). We had to put a personal credit card in there in order to get it started, so that must be removed and another payment method must be put in there.
  - In the Google Cloud Platform under Cloud Build then Triggers, you will be able to connect your GitHub account and repos in order to run the Docker Image. There is already one set up to the GitHub repo CarlyOtt/CIS\_Docker\_Repo. In order to create a new one, just copy the format for that one and connect it to a different GitHub repo.
  - Upon a trigger (pushing/committing to the GitHub repo), a build will automatically start from the CloudBuild.yaml located in the GitHub repo. From there, it will use the Dockerfile that is also located in the GitHub repo and run the Docker image that is located in DockerHub. After the run, the CloudBuild.yaml file will save the

image and create/save the artifacts that were produced from the build. These artifacts are saved in Cloud Storage in the specified folder that the CloudBuild.yaml file lists (can be changed).

- How to add files and save them to the Docker Image:
  - Open two different terminals. One for copying the files in and one for running the Docker image.
    - If you do not have access to the Docker image on a terminal, you can pull it from DockerHub with the command: `sudo docker pull <image name and tag>`. Make sure you are logged into Docker first (`sudo docker login`).
  - In one of the terminals, run the Docker image with the command: `sudo docker run --interactive --tty <Image name>`.
  - In the second terminal, figure out the name of the running container with the command: `sudo docker container ls`. Remember the name of the container running.
  - CD into the directory of the file/files you want to copy into the Docker image.
  - Once there, you can copy the file/files into the Docker image with the command: `sudo docker cp <file name> <container name>:<location/directory you want to save the file in the Docker image>`.
    - Ex) `sudo docker cp Text.txt ContainerName:/home/stk11.5.0/bin`
  - After the files are copied into the Docker image, in the second terminal, you want to stop the running container with the command: `sudo docker container stop <container name>`.
  - Commit the Docker image and choose a new name to call the saved image (you can put the old name and it will overwrite the old image) with this command: `sudo docker commit <Container Name> <new image name>`.
  - To push the new Docker image to DockerHub, use the command: `sudo docker push <image name>`.
  - \*NOTE: If the STK license ever needs to be changed, this is the location of the STK license file that is located in the Docker image:  
/home/user\_info/STK11/Config
- Connecting the Slack bot to report the tests
  - <https://github.com/nathaniel-nemiroff/slackbottest>
  - Using this code as a base, a couple methods of implementation are possible, either through simple HTTP requests, or by pulling directly from the Google Drive API.
  - All methods require the slackclient methods in order to interact with the slack bot
  - Proper authentication is required for most methods, ensure that your script is utilizing the right authentication.

## IX. Conclusions and Future Work



## IX.1.Limitations and Recommendations

The current limitations are that no physical properties simulation occur, as this is outside the scope of STK. We recommend, as mentioned before, to use the reports STK generates about orbital data to feed a simulation in Blender or Solidworks. These tools are designed for heat transfer and many other tests.

## IX.2.Future Work

1. As of right now, our Google Cloud CI is able to build and run our STK engine Docker image. It saves the artifact created during the build in the Google Cloud Storage and the results from the run are printed somewhere in the build log that is generated. Unfortunately, the reports that are generated during a build are not saved as a text file once the build is over.
2. Our Google Cloud CI is unable to run the code that generates the area reports and the power reports (running 'java Main.java' from the AWT\_STK\_X\_GfxAnalysis\_SolarPanelTool code sample). Our dockerfile contains all of the AGI libraries that are needed to run the code, but there is some other library that is missing. When the code is run, an error occurs that says "symbol is missing" which is contained in MainWindow.java. In order to run this code, we used the following commands at the end of our Dockerfile that can be seen in the picture below. This code seems to work. We are just missing something earlier in the Dockerfile, or the Docker image that is needed to run this code. The reports that are generated from this code are called: Sat1\_Power.txt and Sat1\_Area.txt. (See Appendix D for build error picture)

```
WORKDIR /home/stk11.5.0/CodeSamples/CustomApplications/Java/AWT_STK_X_GfxAnalysis_SolarPanelTool/src
RUN ls
RUN java Main.java
WORKDIR /home/stk11.5.0/CodeSamples/CustomApplications/Java/AWT_STK_X_GfxAnalysis_SolarPanelToo
RUN cat Sat1_Area.txt
RUN cat Sat1_Power.txt
```

---

3. STK reports are not human readable at the moment. This data should be parsed and cleaned (using JSON files) to improve structures team's ability to the reports.
4. The Slackbot is unconnected and needs to be properly charged with the task of updating any slack channel with a link to the report.
5. The reports that can be saved off should be moved to a mounted cloud storage link for long term preservation. This stops relying on GCB's storage capabilities.
6. Blender or solidworks physical simulation of physical model of satellite.

## **X. Acknowledgements**

Thanks to Dr. Crandall and Cougs in Space for the opportunity to work on this project. It has been a wild yet informative ride. Also thanks to Steven Dieckmann, who worked diligently in the first semester of the project.

## **XI. Glossary**

AGI	Analytical Graphics, Inc.
CD	Continuous Deployment
CI	Continuous Integration
CiS	Cougs in Space
OSS	Orbital Simulation Suite
STK	Systems Tool Kit
UI	User Interface
VM	Virtual Machine

## **XIII. Appendix A – Team Information**

Carly Ott, Elijah Craig, Jesslyn Khamsoi, and Nathan Nemiroff



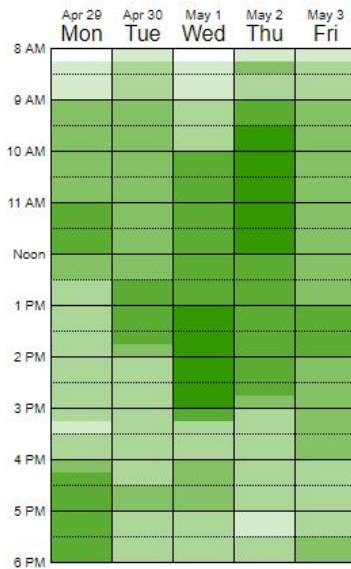
## **XV. Appendix B - Project Management**

- Used Facebook Messenger to communicate
- Used When2meet.com to figure out times to meet
- We would meet whenever we had a report or something else due. We would also meet whenever someone really needed help from other people from the team on what they were working on.

## Group's Availability

1/6 Available  6/6 Available

Mouseover the Calendar to See Who Is Available



CarlyOtt / CIS\_Docker\_Repo
Watch 1
Star 0
Fork 0

[Code](#)
[Issues 0](#)
[Pull requests 0](#)
[Projects 0](#)
[Wiki](#)
[Insights](#)
[Settings](#)

No description, website, or topics provided. [Edit](#)

Manage topics

127 commits
2 branches
0 releases
2 contributors

Branch: master
New pull request
Create new file
Upload files
Find File
Clone or download

CarlyOtt
Update cloudbuild.yaml
Latest commit 26d7680 an hour ago

Dockerfile

Update Dockerfile

7 hours ago

README

Update README

17 days ago

cloudbuild.yaml

Update cloudbuild.yaml

an hour ago

extra.txt

Create extra.txt

7 hours ago

test1

Update test1


13 days ago


README

This is a testing repo for our dockerfile.  
Upon a trigger (like pushing), it will connect with our Dockerhub account and will force an automatic build there with the Dockerfile associated with this repo.

# XVI. Appendix C - Docker/CI Images

## DockerHub:

 carlyishere / stk\_cis

STK IMAGE FOR DOCKERFILE 

Last pushed: 15 hours ago

Docker commands




To push a new tag to this repository.

`docker push carlyishere/stk_cis:tagname`

Public View


Tags

This repository contains 3 tag(s).

final		15 hours ago
updated2		3 days ago
updated1		3 days ago

See all

Recent builds

 CarlyOTT/423dockertest

Build in 'Docker1' (b8fa...
Build in 'Docker1' (5c24...
Build in 'Docker1' (5c24...
Build in 'Docker1' (5c24...
Build in 'Docker1' (ee59...

## Dockerfile:

```
FROM carlyishere/stk_cis:final

ARG LDLIBRARYPATH=/home/stk11.5.0/bin:$LDLIBRARYPATH
ENV LD_LIBRARY_PATH=$LDLIBRARYPATH
ARG STKINSTALLDIR=/home/stk11.5.0
ENV STK_INSTALL_DIR=$STKINSTALLDIR
ARG STKCONFIGDIR=/home/user_info/STK11/Config
ENV STK_CONFIG_DIR=$STKCONFIGDIR
ARG JAVAHOME=/usr/lib/jvm/jdk-12.0.1
ENV JAVA_HOME=$JAVAHOME

ARG ANTHOME=/home/ANT/ANT
ENV ANT_HOME=$ANTHOME
ARG PATHH=${ANT_HOME}/bin:${JAVA_HOME}/bin:${PATH}
ENV PATH=$PATHH

ARG CLASS_PATH=.
ARG CLASS_PATH=${CLASS_PATH}:/home/stk11.5.0/bin/agi.stk.javadocs.jar
ARG CLASS_PATH=${CLASS_PATH}:/home/stk11.5.0/bin/agi.swing.jar
ARG CLASS_PATH=${CLASS_PATH}:/home/stk11.5.0/bin/agi.swt.jar
ARG CLASS_PATH=${CLASS_PATH}:/home/stk11.5.0/bin/agi.core.jar
ARG CLASS_PATH=${CLASS_PATH}:/home/stk11.5.0/bin/agi.core.swing.jar
ARG CLASS_PATH=${CLASS_PATH}:/home/stk11.5.0/bin/agi.core.swt.jar

WORKDIR /home/OrbitalSimulationSuite/NoGraphics
RUN ls
RUN ant run >> txt.txt
RUN cat txt.txt
```

## CloudBuild.yaml

```
steps:
- name: 'gcr.io/cloud-builders/docker'
  id: runDockerfile
  args:
  - 'build'
  - '-t'
  - 'gcr.io/cpts423seniorproject/github.com/carlyott/423dockertest:latest'
  - '--file=Dockerfile'
  - '.'
tags: ['testing1.2']
images: ['gcr.io/cpts423seniorproject/github.com/carlyott/423dockertest:latest']
```

## Successful Build Example:

### Build information

Status	✔ Build successful
Build id	be5f6ec7-8ef6-4033-bf08-50732d80b154
Image	gcr.io/cpts423seniorproject/github.com/carlyott/423dockertest:latest
Artifacts	6 (View manifest)
Trigger	Push to master branch (Push to any branch)
Source	GitHub CarlyOtt/423dockertest <a href="#">↗</a>
Git commit	5a7cfe1bcc4a24276aa11bda1df9809ff30c69d0 <a href="#">↗</a>
Started	May 1, 2019 at 11:35:05 AM UTC-7
Duration	4 min 15 sec

### Build steps

✔ runDockerfile	3 min 39 sec ▾
gcr.io/cloud-builders/docker — build -t gcr.io/cpts423seniorproject/github.com/carlyott/423dockertest:latest -f=Dockerfile.	

## Unsuccessful Build Example:

✔ copyFiles

gcr.io/cloud-builders/docker — container ls

1.0 sec ▾

❌ gcr.io/cloud-builders/gutil

'ls /wd'

3 sec ▾

Logs

Download logs

✔ Show newest logs first  
✔ Wrap lines

ERROR: build step 3 "gcr.io/cloud-builders/gutil" failed: exit status 1  
ERROR  
Finished Step #3  
Step #3: CommandException: Invalid command "ls /wd".  
Step #3: Already have image (with digest): gcr.io/cloud-builders/gutil  
Starting Step #3  
Finished Step #2 - "copyFiles"  
Step #2 - "copyFiles": 80acc9836a64 gcr.io/cloud-builders/metadata "/metadata" 5 minutes ago Up 5 minutes metadata  
Step #2 - "copyFiles": c612044d10b af2774c517aa "ash" 5 minutes ago Up 5 minutes docker\_token\_container  
Step #2 - "copyFiles": 8059fbad73ef gcr.io/cloud-builders/docker "/usr/bin/docker con..." 1 second ago Up Less than a second step\_2  
Step #2 - "copyFiles": CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES  
Step #2 - "copyFiles": Already have image (with digest): gcr.io/cloud-builders/docker  
Starting Step #2 - "copyFiles"  
Finished Step #1 - "runDockerfile"  
Step #1 - "runDockerfile": Successfully tagged gcr.io/cpts423seniorproject/github.com/carlyott/423dockertest:latest  
Step #1 - "runDockerfile": Successfully built d4165dab63a9  
Step #1 - "runDockerfile": ----> d4165dab63a9  
Step #1 - "runDockerfile": Removing intermediate container ada9d26d3ach  
Step #1 - "runDockerfile": Total time: 4 seconds  
Step #1 - "runDockerfile": BUILD SUCCESSFUL  
Step #1 - "runDockerfile": run:  
Step #1 - "runDockerfile":

## Area & Power Reports Build Error:

```
ERROR: build step 0 "gcr.io/cloud-builders/docker" failed: exit status 1
ERROR
The command '/bin/sh -c java Main.java' returned a non-zero code: 1
error: compilation failed
2 errors
location: class Main
symbol: class MainWindow
^
MainWindow mw = new MainWindow();
Main.java:29: error: cannot find symbol
location: class Main
symbol: class MainWindow
^
MainWindow mw = new MainWindow();
Main.java:29: error: cannot find symbol
---> Running in cdd2fcf097bb
Step 55/58 : RUN java Main.java
```

## **XVII. Appendix D - Instructions for future workers**

For the sake of continuous integration, you will most likely need to choose a linux distro. The ideal choices for this are CentOS (Based on RedHat), or Ubuntu. We chose Ubuntu 18.04 and believe this to be the right choice. This leads to an issue with STK though, as the OnDemand type license provided by STK through Educational Alliance Program is incompatible with all Linux distros. You will need NodeLocked type licenses. This will require sending you computer's MAC Address to the future mentor, probably Dr. Crandall, so he can request a NodeLocked Education license for each of your machines including the VM you will use.

The education license may not suffice to your needs, and if that arises consult your mentor. We had an issue of Java objects that had functions that were locked behind a paywall saying you need to upgrade to STK Pro.

You will also want to use Eclipse and set up the code samples following the guide<sup>1</sup> found at Development Environments/AGI Java API/Development Environment/Eclipse Environment/Configuring JavaSamples. This is crucial, as this will be a great deal of the small amount of documentation you will have access to.

You will also want to install STK with the guide<sup>2</sup> at the link provided.

When it comes to the delivering of the report, see that the cloud storage solution provides an ability to mount to your system. Google Drive has this type of functionality, and this will be more accessible and readable than GCB's system.

---

<sup>1</sup> <http://help.agi.com/stkdevkit/index.htm>

<sup>2</sup> <http://help.agi.com/stkEngineOnUNIX/index.htm>