Affan Atif
CSC 4999
Honors Thesis
# Table of Contents

Table of Figures .................................................. 2
Background ......................................................... 3
Tools Used .......................................................... 3
Database of Objects ............................................... 4
GPU Percentile Program .......................................... 6
  Concept ......................................................... 6
  Algorithm Development ....................................... 6
  Analysis ........................................................ 7
Multivariate Degradation Regression .......................... 8
  Concept ......................................................... 8
  Algorithm Development ....................................... 9
  Analysis ........................................................ 11
GPU Percentage Regression .................................... 16
  Concept ......................................................... 16
  Algorithm Development ..................................... 17
  Analysis ........................................................ 18
Conclusion ......................................................... 21
References ........................................................ 23
# Table of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1.0</td>
<td>5</td>
</tr>
<tr>
<td>Figure 2.0</td>
<td>5</td>
</tr>
<tr>
<td>Figure 3.0</td>
<td>5</td>
</tr>
<tr>
<td>Figure 4.0</td>
<td>7</td>
</tr>
<tr>
<td>Figure 5.0</td>
<td>8</td>
</tr>
<tr>
<td>Figure 6.0</td>
<td>8</td>
</tr>
<tr>
<td>Figure 7.0</td>
<td>9</td>
</tr>
<tr>
<td>Figure 8.0</td>
<td>10</td>
</tr>
<tr>
<td>Figure 9.0</td>
<td>11</td>
</tr>
<tr>
<td>Figure 10.0</td>
<td>12</td>
</tr>
<tr>
<td>Figure 11.0</td>
<td>12</td>
</tr>
<tr>
<td>Figure 12.0</td>
<td>13</td>
</tr>
<tr>
<td>Figure 13.0</td>
<td>13</td>
</tr>
<tr>
<td>Figure 14.0</td>
<td>14</td>
</tr>
<tr>
<td>Figure 15.0</td>
<td>15</td>
</tr>
<tr>
<td>Figure 16.0</td>
<td>16</td>
</tr>
<tr>
<td>Figure 17.0</td>
<td>18</td>
</tr>
<tr>
<td>Figure 18.0</td>
<td>18</td>
</tr>
<tr>
<td>Figure 19.0</td>
<td>19</td>
</tr>
<tr>
<td>Figure 20.0</td>
<td>19</td>
</tr>
<tr>
<td>Figure 21.0</td>
<td>20</td>
</tr>
<tr>
<td>Figure 22.0</td>
<td>20</td>
</tr>
</tbody>
</table>
Background

Today’s electronic devices are capable of computations and calculations that outmatch those that came before them. More fields around the world are relying on this technology to automate and improve their analytics for improved efficiency. Every year, technology is evolving to increasingly handle the needs of an interconnected society. However, this increased performance comes at a cost: reduced energy efficiency. The increased performance relies on increased energy and battery consumption within those devices. Because of this increased need for energy, batteries have been evolving to meet the needs of demanding technology in a sustainable way.

Although the sizes of batteries and the amount of energy they can contain have been increasing over the past few decades, the challenge still continues to find ways to make their performance more efficient to allow them to keep powering technology. For example, smart phones can dedicate many of their processing tasks to other shared devices such as smart watches or smart speakers, ultimately increasing the battery life of all the shared devices. The increased energy requirements has prompted creative solutions to provide enough battery power while maintaining performance.

Currently, the Energy Aware Autonomous Systems Lab, led by Dr. Marco Brocanelli, aims to tackle this issue of efficient energy through the optimization of the coordination between the sensing resources and distributed autonomous components. This method attempts to find the most optimal way to distribute resources throughout the device while also using as minimal energy as possible required. The EAS-Lab’s main focus surrounds the following areas: 1. Cyber Physical Systems 2. Energy aware systems 3. Internet of Things 4. Edge Computing and 5. Embedded and real time systems (EAS-Lab, 2020).

The EAS Lab currently utilizes smartphone technology as well as its augmented reality functionalities to simulate three dimensional objects in real time environments. These simulations can help provide valuable insights on how smartphone performance can be related to the GPU performance. The lab focuses on utilizing creative approaches to analyze energy performance and ways to make it more efficient.

The focus of the thesis will be to discuss the various projects completed and its results to help support the analytics which will be used by the EAS-Lab to identify areas of improvement and accuracy of current test results.

Tools Used

In order to properly simulate the computing process for smartphones, the EAS-Lab utilized a graphical modeling toolset called “Blender”. Blender is an open source animation, simulation, and rendering software that can be used to create realistic 3D Objects (Blender, 2020). These objects can be used to analyze rendering times on smartphones as well as the effect of degradation and its relation to GPU usage. This software will be used on the Windows and Linux platforms to create an initial database of 50 realistic objects for further testing.

For analysis of GPU usage in relation to object density and mesh count, the programming language Java was used. The compiler used was IntelliJ for the Windows platform.
To create the error values for each degradation ratio, the programming language Python was used. The reason for this selection was for its capability for modeling and graphing. The compiler used was Spyder for the Windows platform. Additionally, python was used to create Blender scripts. Blender has built in support for python where its features can be accessed programmatically instead of through a GUI. This is advantageous because of the fact that the script can be run multiple times for each object at a much faster rate compared to running it manually. Blender has its own API that can be used through python and was used to retrieve the GPU and degradation values for several objects.

For analysis of the degradation errors resulting from applying the degradation errors, Microsoft Excel was used. The reason for this was for the software’s ability to calculate various statistical values such as standard deviation. Additionally, the software was used to graph and compare predicted and actual values for various variables.

These tools were used throughout the duration of the EAS-Lab. They helped to create and analyze the various calculations conducted for the purpose of analyzing the effects of 3D Objects on a mobile device. These results will be used for developing solutions for increased energy efficiency on mobile devices.

**Database of Objects**

In order to conduct the various experiments regarding energy efficiency, 3D Objects were utilized to simulate mobile device renderings through the Blender software. These objects were based on real life items such as vehicles, trees, furniture, and handheld devices. These types of objects were selected based on the fact they replicate everyday objects that could be used by an augmented reality application on a mobile device.

The Blender software models an item based on triangles. There can be between 1 and many millions of triangles that comprise the physical structure of an object. A relatively low number of triangles indicates a low quality object with not as many distinct features, whereas an object with many triangles indicates an object with many distinct features and easily distinguishable. There is a direct correlation between the number of triangles in an object and the processing time required to render that object on the screen. For example, an object with 1,000,000 triangles will result in a longer rendering time compared to an object with 500,000 triangles. The number of triangles are the backbone of how an object is created and modeled in blender and serves as the basis to many of the calculations and experiments done in the EAS-Lab.

In order to begin with the experiments, an initial database of high quality objects were required. As part of the EAS-Lab, a database was created with 50 objects. Each object is unique and features a different number of triangles that comprise it. Additionally, each object is a replica of many real life items that a person might encounter in their day to day lives. This was done to ensure that the calculations and analysis are done using objects that are realistically encountered day to day and could appear on an augmented reality application on a smartphone.
The objects were retrieved with only their physical shapes without their textures. The original object was saved as a .obj file which is interpreted by Blender as a mesh of triangles that form the structure of the object. In order to apply the colors and the various textures of each object, the .mtl file was applied through blender to display the final object. The .mtl files are simply notepad files that contain link references to the location of the textures on the computer system, which are saved as JPEG images. The .mtl files were updated to contain file references on as many surfaces that were blank. Those references are retrieved and the images are applied to the respective surfaces.

For the curated database of objects, the textures were applied to each object and combined into each of their own folders for ease of reference. The resulting object after application of the proper textures is as follows:
Additionally, the objects are most effective for analysis when they are of life size dimensions. Initially, many of the curated objects were of unrealistic dimensions on blender and varied widely for different types of objects. In order to normalize the objects, each object was compared with real-life counterparts to determine an average size of the length, width, and height dimensions of the object. A script was used to apply these dimensions to resize the object and re-save them with the proper materials applied. This process was repeated for each of the 50 objects. Once this was completed, the database was officially normalized and ready for use in analysis.

GPU Percentile Program

Concept
In order to understand the effect of the number of triangles on the GPU, it is important to analyze the change in GPU utilization percentage over certain durations of time. This analysis will help determine the significance of the correlation between object rendering and how much strength is required from the GPU. In order to accomplish this, an initial blender script was executed to collect GPU data on a test object. The script was run for a certain amount of time, and for each interval of about 12 seconds, the number of triangles was recorded. Additionally, for about each second of the script execution, the current GPU utilization was also recorded. The exact times for each of these values was recorded down to the exact millisecond to ensure accurate comparison. These values were saved into two separate notepad files after execution, the first file holding the number of triangles and time, and the second holding the GPU utilization and time.

Algorithm Development
The algorithm to analyze GPU utilization was written using Java. Initially, the values for time and number of triangles were read into Java from notepad using the input/output library. Next, each line of the file was split into separate indexes of an array based on whitespace. This is because the values for date, time and number of triangles were saved together on the same line. Additionally, the time itself was also split by the colon character. The values for hour, minute were converted to seconds and added to their respective seconds value to get the final time value in seconds, which later on allows for easier comparison between time values. The triangles values were also saved into its own double variable. In order to maintain consistency between the time in seconds and the number of triangles, a common counter variable was used in order to place the values into the correct index of their respective arrays. This process was repeated for each line in the text file until all the values were parsed and properly sorted through.

A similar process began for the next text file which contained the date, time and the current GPU utilization percentage. This time, however, the time value and its respective GPU values were located on adjacent lines in the file. The current time value was displayed first and the corresponding GPU percentage was located directly on the next line. The previous method of each line belonging to its own time value was no longer valid. Therefore, the file was traversed to carefully check which values were being interacted with. A row counter variable was used to keep track of which line was being read. If the row value was an even value, the row contained a time value and thus the time value was split based on
the colon character and converted into a single value based on seconds and placed into a new array. If the row value was an odd value, the row contained the GPU percentage. The line was split based on whitespace character and the first index of the row (percentage) was placed into its own array. In order to maintain consistency, two counter variables were used for both even and odd rows and allowed for correct assignment for the time and GPU percentage to their respective arrays. This process was repeated for each line in the text file until all the values were parsed and properly sorted through.

After all the values were properly organized into the correct variables, the process to calculate mean GPU utilization percentage could be conducted. For the sake of clarity, the first text file containing the time values will be referenced as “File A”, and the second text file containing the time values will be referenced as “File B”. Initially, a new double variable was declared which will record the sum total of GPU percentiles called “sum”. A for loop was initialized to traverse each time value from File A. A while loop was also declared within the for loop to traverse each time value from File B. Therefore, each time value from File A will be compared with each time value from File B. The current time value being iterated has 2 seconds added to it to account for the time taken to begin the rendering process. During the execution of the while loop, each iteration will add the current GPU percentage to the “sum” variable. However, if the time value from File B is between the time value from File A and time value from File A + two, then the number of iterations is saved and is used to divide the sum variable. The resulting value will be the average GPU percentage over those iterations. The value is output to the screen along with the time, number of triangles, and mean GPU. The values of the variables sum, iteration, and mean GPU are then reset, and the next iteration of the for loop will continue and the process is repeated until the end of the file.

![Figure 4.0. Sample Output for GPU Utilization](image)

**Analysis**

After the program finishes execution, the user is displayed all of the GPU values and the corresponding number of triangles for that interval. In terms of GPU utilization, the more triangles that are being processed in the Blender object, the higher the average GPU utilization. Conversely, the lower the triangle count, the lower the average GPU percentage. According to figure 4.0, at time 00:00:00:0000 with 0 triangles, which represents the start of the blender script, the average GPU utilization percentage is 34.33%. In the next interval, when the number of triangles increases to 1000, the average GPU utilization rises to 54.07%. This increase in triangles contributes to a direct increase in utilization. Similarly, the
following interval contains 500 triangles, a decrease of 500 from the previous interval, and exhibits a
decrease in average percentage to 51.88%. This trend continues throughout the output.

Based on these results, it is shown that the average GPU utilization percentage is directly correlated to an
increase/decrease in triangles. The decimation of an object leads to an overall decrease in GPU percentage
and vice-versa.

**Multivariate Degradation Regression**

**Concept**

In order to analyze the result of a degradation for an object, it is essential to calculate the values resulting
from a decimation action while taking into account the distance from that object. A decimation filter can
be applied many times with different ratios, resulting in vastly different degradation errors. These values
are also dependent on the gamma values which can be calculated in alternate ways. Therefore, in order to
analyze the degradation errors at different ratios and gamma values, it is critical to be able to create a
model that can predict those error values for any object.

\[
dy = a(r^2) + b(r) + \frac{c}{d^y}
\]

*Figure 5.0. The original formula for degradation error.*

The polynomial formula from figure 5.0 refers to the calculation of the degradation error. The variables
‘a’, ‘b’, and ‘c’ refer to the coefficients that are determined through calculation. The variable ‘r’ refers to
the decimation ratio, ‘d’ refers to the distance of the object, and ‘y’ refers to the gamma value. The values
for ‘a’, ‘b’, and ‘c’ will vary for each object.

In order to accomplish the prediction of values, a regression must be utilized to generate a model. Because
there are multiple variables that can alter the outcome of the degradation formula coefficients, a
polynomial regression would initially be a favorable type of model. However, a linear regression is more
optimal because the original formula for degradation can be rewritten to form a linear equation, creating a
more accurate model. Therefore, the equation must be transformed to fit a linear expression.

The rewrite of the polynomial formula is as follows. The \(d^y\) can be removed from the denominator and
represented as a new variable called ‘z3’, and can be applied to each term of the numerator instead as a
multiplication factor. The resulting formula becomes:

\[
z3 \cdot (a \cdot (r^2)) + z3 \cdot (b \cdot (r)) + z3 \cdot c
\]

*Figure 6.0. The degradation formula without the denominator.*

The formula can be simplified further. The terms ‘z3’ and ‘r^2’ from the first quantity in figure 6.0 can be
combined to create a new variable ‘z1’. Similarly, the terms ‘z3’ and ‘r’ can again be combined to create a
new variable ‘z2’. Because there is no ‘r’ term in the third quantity of the formula with ‘c’, the last quantity is unchanged. The further updated version of the formula becomes:

\[(z_1 * a) + (z_2 * b) + (z_3 * c)\]

Figure 7.0. The final linear form of the degradation formula.

The new formula from figure 7.0 is the same as the original formula from figure 5.0, however three new variables ‘z1’, ‘z2’, and ‘z3’ were created to hold the ratio and denominator terms, which transforms the equation into linear form without any terms with a degree higher than 1. This new formula can now be used to utilize linear regression and will generate predicted values for ‘a’, ‘b’, and ‘c’.

The value for gamma will also vary for the degradation model. This is because there are multiple methods to calculate the gamma value for the model. In particular, this analysis focused on three versions of the gamma value. The first method calculates gamma by taking the logarithm of the first index degradation value divided by the second index degradation value, and dividing it by the logarithm of the second index distance value divided by the first index distance value. The second method to calculate gamma involves taking the logarithm of the first index degradation value divided by the final index degradation value, and dividing it by the logarithm of the final index distance value divided by the first index distance value. The third method to calculate gamma is simply taking the average of the first two gamma values. These three methods will produce three different gamma values that will affect the multivariate regression, leading to different coefficient values for ‘a’, ‘b’, and ‘c’. These sets of coefficients will then be compared with each other to determine the most accurate coefficient values.

To create the model for degradation errors, a dataset containing the values for distances and ratios was required. These values were collected by running a blender script through python on a specific blender object. The amount of distance values as well as the range between them can vary between each object. Additionally, there can be any number of ratio values to be applied for the degradation in the file. Using these values, the polynomial regression will output the predicted coefficient values to generate a formula that will most resemble the actual degradation graph for a particular object.

Algorithm Development

The algorithm to calculate predicted degradation values was written using python. The ‘sklearn’, ‘pandas’, ‘numpy’, ‘matplotlib’, ‘statistics’, and ‘math’ libraries were imported to assist with the model calculation and graphing. Initially, the csv file containing the distances and ratio values are read into the script using the pandas library. The row headers are counted and saved into a variable and list respectively. The row headers contain the distance values in the first column, and the ratio values are followed after. This will help determine which ratios are being used for the particular object. The script then reads the values in each column iteratively, and appends each distance value and ratio degradation value into their own list variables separate from the row header lists using for loops. The list containing the degradation values is called ‘Y’. The gamma values are calculated using the degradation values and the exact formulas for gamma are discussed in the previous section. The average gamma value is also calculated by averaging the previous two gamma values. These values are appended to the gamma list
variable. At this point, the necessary values have been organized into the proper variables and are prepared for further calculations.

Three list variables are declared for the ‘z1’, ‘z2’, and ‘z3’ terms. A for loop is initialized to iterate through the ratio values. Within the for loop, another for loop is initialized to iterate through the distance values in order to calculate the ‘z3’ term by dividing 1 by the current iteration’s distance value to the power of the current gamma value. These values are appended to the z3 list. Similarly, Another for loop is initialized to iterate through the z3 list to calculate the z1 term by multiplying the current iteration’s z3 value with the square of the current ratio value. These values are appended to the z1 list. A third and final for loop is initialized to iterate through the z3 list to calculate the z2 term by multiplying the current iteration’s z3 term with the current ratio value. These values are appended to the z2 list. At this point, the ‘z1’, ‘z2’, and ‘z3’ values have been determined for the specified gamma value. A final three dimensional list called ‘X’ is initialized in which the values for z1, z2, and z3 are appended accordingly into. For each iteration of the ratio values, the values for z1, z2, and z3 based on ratio are appended into the same X list.

Next, the X and Y lists are transposed to fit a one dimensional list, and are then input into the linear regression function. The function returns a multidimensional list of coefficients that can be iterated through. In figure 8.0, the value in index 0 refers to coefficient ‘a’, index 1 refers to coefficient ‘b’, and index 2 refers to coefficient ‘c’. These coefficients are the predicted values that can be inserted into the original degradation formula to predict the degradation values at any ratio. This entire algorithm is repeated two more times however with the two different gamma values. The resulting output is three lines with each line displaying the recommended coefficient values based on a different gamma value.

```
Multivariate Linear Regression Coefficients: [0.3208, -0.4145, 0.1538]
Multivariate Linear Regression Coefficients: [0.3558, -0.4608, 0.1715]
Multivariate Linear Regression Coefficients: [0.3502, -0.4532, 0.1684]
```

*Figure 8.0. Sample output for Multivariate Linear Regression*

Following the display of the coefficients, these values are saved into a three dimensional list. Each row in the list represents the model derived from a certain gamma value. A for loop is initialized to iterate through the multidimensional list. On each iteration of the for loop, one set of the ‘a’, ‘b’, and ‘c’ values are selected and used in the degradation formula. The original distance values and the corresponding gamma value are also selected to maintain consistency in the formula. The resulting predicted degradation values are compared with the actual degradation values, and using summation, the root mean squared errors are calculated for each ratio value. The values are output to the screen and will be between 0 and 1. The closer the error is to 0, the more accurate the model is whereas the closer to 1 the value is, the more inaccurate the model is. A float variable is initialized to calculate the average RMSE error. Each ratio’s RMSE value for a particular gamma value is added and then divided by the number of rows to calculate the average RMSE error. This process is repeated for each set of coefficients and the average RMSEs are displayed. The set with the lowest average RMSE is determined to be the most accurate set of coefficients for the degradation model. Therefore, the gamma value that belongs to the lowest RMSE set is copied and the degradation formula is executed one last time to produce the final coefficients. This final set of the ‘a’, ‘b’, ‘c’, and gamma value coefficients are estimated to be the most accurate values for the degradation formula for that specific object.
Analysis
The values for ‘a’, ‘b’, and ‘c’ are determined by the distance, ratios, and degradation values, and the gamma for the multivariate regression. Using these coefficients, the model can predict the degradation errors for any ratio to a certain accuracy depending on the object being analyzed. The RMSE errors indicate the accuracy of the model and the lower values indicate more accurate coefficients. Ultimately, this model will allow for accurate analysis to how the number of triangles in an object relate to the ratio of decimation, distance from the object, and the gamma values.

In order to conduct the regression, several blender objects were utilized. Each object had its own unique structure as well as different triangle counts. The objects are as follows: Andy, Chair, Plant, and Plane. The following are depictions of these objects in their 3D form:

![Objects Image](image)

*Figure 9.0. Plant degradation graphs for 0.2 and 0.8 with Gamma = 0.9926102*
Figure 10.0. Plant degradation graphs for 0.2 and 0.8 with Gamma = 1.29323

Figure 11.0. Chair degradation graphs for 0.2 and 0.8 with Gamma = 0.918192
Figure 12.0. Chair degradation graphs for 0.2 and 0.8 with $\Gamma = 1.026149$

Figure 13.0. Andy degradation graphs for 0.2 and 0.8 ratios with $\Gamma = 0.905601731$
Figure 14.0. Plane degradation graphs for 0.2 and 0.8 ratios with Gamma = 1.324801
In figure 9.0, the yellow line represents the actual degradation values for a 0.2 ratio for the plant object. The green line represents the predicted degradation values for a 0.2 ratio. After finding the coefficient values for ‘a’, ‘b’, ‘c’, and gamma, the formula is created and the distance values are inputted to retrieve the predicted degradation errors. The result is the above graph in which both of the trend lines are mostly similar. Additionally, this graph was calculated using the first gamma value of 0.9926102, which was determined using the first gamma equation outlined previously. In comparison, the graph in figure 10.0 was calculated using the second gamma equation, resulting in 1.29323. The differences between the two graphs can be viewed. Notably, the Model 0.2 graph line trends higher than the original 0.2 line in figure 9.0, whereas it trends closer in figure 10.0. After running the script, the most optimal gamma was determined to be 1.29323. From here, the RMSE can be calculated to determine the accuracy of the model. For Gamma value 0.9926102, the average RMSE is 0.00319, whereas for Gamma value 1.29323, the average RMSE is 0.00239. Using the derived coefficients, degradation values can be collected for any distance and ratios with reasonable accuracy.

In figure 11.0, the first gamma value was calculated to be 0.918192. In comparison, the graph in figure 10.0 was calculated using the second gamma equation, resulting in 1.026149. The differences between the two graphs can be viewed. All other coefficients were held constant. Notably, in figure 12.0, the Model 0.2 and Model 0.8 graphs both trend closer to the original 0.2 and 0.8 compared to figure 11.0. This was because the most optimal gamma was determined to be 1.026149. From here, the RMSE can be calculated to determine the accuracy of the model. For Gamma value 0.918192, the average RMSE is 0.0033, whereas for Gamma value 1.026149, the average RMSE is 0.00168.
In figure 13.0, the original graphs for ratios 0.2 and 0.8 as well as the Model graphs for 0.2 and 0.8 are shown. The graphs for Model 0.8 and original 0.8 are most similar to each other compared to the other lines. The gamma values were the same for both the first and second equation where gamma = 0.905601. The average RMSE is 0.001213.

In figure 14.0, the first gamma value was calculated to be 1.324801. In comparison, the graph in figure 15.0 was calculated using the second gamma equation, resulting in 1.51285039. The differences between the two graphs can be viewed. All other coefficients were held constant. Notably, in figure 15.0, the Model 0.8 graph trended more accurately to the original 0.8 graph compared to the graph in figure 14.0. This was because the most optimal gamma was determined to be 1.51285039. From here, the RMSE can be calculated to determine the accuracy of the model. For Gamma value 1.324801, the average RMSE is 0.001802, whereas for Gamma value 1.51285039, the average RMSE is 0.00219.

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Coefficient A</th>
<th>Coefficient B</th>
<th>Coefficient C</th>
<th>Gamma</th>
</tr>
</thead>
<tbody>
<tr>
<td>Andy</td>
<td>0.1149</td>
<td>-0.1819</td>
<td>0.0935</td>
<td>0.905601731</td>
</tr>
<tr>
<td>Chair</td>
<td>-0.0463</td>
<td>0.0252</td>
<td>0.0484</td>
<td>1.026149</td>
</tr>
<tr>
<td>Plane</td>
<td>0.2635</td>
<td>-0.3629</td>
<td>0.159</td>
<td>1.324801</td>
</tr>
<tr>
<td>Plant</td>
<td>0.3502</td>
<td>-0.4532</td>
<td>0.1684</td>
<td>1.29323</td>
</tr>
</tbody>
</table>

*Figure 16.0. The final derived coefficients and the calculated optimal gamma value for each object.*

**GPU Percentage Regression**

**Concept**

It is important to understand the relationship between the triangles that form an object and the GPU utilization percentage. As the number of triangles increases for an object, the resources used to render the object increases as well. The rendering is performed usually by the GPU on most devices. Similarly, on a smartphone, the rendering of augmented reality objects is also done by the phone’s GPU. Therefore, the number of triangles is directly correlated to the GPU utilization percentage. However, the exact ratio for which the percentage changes based on triangles can vary per object. Additionally, the percentage can change based on the distance of the object as well.

When there are no objects currently rendering, the GPU of the device will not necessarily be at 0%. There are other system processes and functionalities that will require the GPU to process. Therefore, the GPU utilization will be at a certain percentage above 0%. Once an object is being rendered, the percentage will increase. However, the percentage will also never increase past 100%. This is because the percentage will
flatten out past a certain number of triangles, dependent on the specific object. When graphed, this creates an interesting trend. The graph line can essentially be divided into three zones. The first zone represents the GPU utilization at “rest” and is depicted by a line with a slope close to 0. The second zone represents the increasing GPU utilization after an object starts rendering and is depicted by a line with a linear slope. The third zone represents the flattening of the GPU utilization after the rendering is complete and is depicted by a line with a slope close to 0. These zones on a graph can help identify the start and end indexes for GPU utilization for an object. Additionally, the second zone’s trend line can be used to calculate a model to help determine a formula that can be applied to all objects instead of one.

Based on multiple GPU utilization graphs for multiple blender objects, the second zone was determined to begin with the index containing the lowest GPU percentage value. Similarly, the second zone was determined to end with the index containing the highest GPU percentage value. Therefore, an algorithm can be developed to detect the maximum and minimum percentage values within the graph, which would indicate the starting and ending index values respectively. These values can be used to isolate the second zone of the graph in order to calculate a linear regression.

In order to calculate a regression model for GPU utilization, a blender script was executed to calculate the percentage utilization through its rendering. Each time label was recorded as well as the number of triangles. The values were saved into a csv file into separate columns.

**Algorithm Development**

The algorithm to determine the three zones of GPU utilization and calculate the regression model was written using python. The ‘pandas’, ‘sklearn’, ‘numpy’, and ‘matplotlib’ libraries were used. After running the blender script to save the percentage and triangle values, the values are read from the csv file and into the python script. The triangle values are saved into a list variable and the percentage values are saved into a corresponding separate list variable. Because the length of the two lists are the same, only one float variable is required to save the length of the list. An additional list called ‘temp’ is declared and the percentage values are deep-copied into this list. The first 5 index values of the new list are removed because the starting values are not indicative of the rest of the graph and can incorrectly alter the outcome of the final indexes. A float variable called ‘min’ is declared and is set equal to the minimum value of the temp array. The function to locate the minimum value is a built in function in python. Next, a for loop is initialized which traverses each value in the temp array and if the current value is equal to the min variable, the current iteration is saved into a counter variable, which displays to the user the index that the minimum value was found. Another float variable called ‘max’ is declared and is set to the maximum value of the temp array. Another for loop is declared which traverses each value in the temp array and if the current value is equal to the max variable, the current iteration is saved into a counter variable and displays to the user the index that the maximum value was found. At this point, the start and end indexes for the second zone have been determined through the min and max variables.

The original lists holding the original triangle values and percentage values are sliced using a for loop which appends the list values for the second zone of the graph into two new lists called ‘X’ and ‘Y’ respectively. At this point, ‘X’ and ‘Y’ are holding the values for the second zone only and can now be used to create a model. The list values are separated into two sets, one set holding 80% of the values and the other holding 20% of the values. The set holding 80% of the values will be used as the training set to
help the model determine the coefficients for the linear regression whereas the set holding 20% of the values will be used as the test set to evaluate the accuracy of the regression. The model is calculated using the training set and the y-intercept as well as the slope coefficient are displayed to the user. Additionally, the RMSE, mean squared error, and mean absolute error are calculated based on the test set and are also displayed to the user. The slope value and y-intercept can be used to create a linear formula that models the second zone of the GPU utilization graph for that specific object.

Analysis

Figure 17.0. Output of the GPU utilization percentage regression values for the Big Cabin Object

Figure 18.0. Graph of the GPU utilization percentage regression values for the Big Cabin Object
Figure 19.0. Output of the GPU utilization percentage regression values for the Small Cabin Object

Figure 20.0. Graph of the GPU utilization percentage regression values for the Small Cabin Object
In figure 16.0, the initial index represents the index at which the second zone of the graph begins. The final index represents the index at which the second zone ends. The intercept and slope values represent the coefficients for linear equations. These values were determined by the initial set of GPU percentage and triangle values for the second zone of the graph only. The first zone and third zone are not representative of the GPU percentage and were thus not used for the regression. The RMSE values indicate the accuracy of the model based on the training set.

For figure 17.0, the starting index is depicted by the first vertical red line at x = 35. The ending index is depicted by the vertical red line at x = 293. These lines represent the boundaries for the starting and ending values for zone 2 of the graph. The points that are located in this zone are used to calculate the regression model. In this case, the slope is determined to be 7.0028226e-05 and the intercept is 43.03711816. Additionally, zones 1 and zones 3 are also labeled however the values from these zones are not used for the regression calculation. The values in zone 3 indicate a flattening out percentage of 97% with no increase.
For figure 19.0, the starting index is depicted by the first vertical red line at \( x = 11 \). The ending index is depicted by the vertical red line at \( x = 154 \). In this case, the slope is determined to be 0.00012216 and the intercept is 47.51105265. The majority of the values for the graph are located in zone 2.

For figure 21.0, the starting index is depicted by the first vertical red line at \( x = 15 \). The ending index is depicted by the vertical red line at \( x = 83 \). The slope is calculated to be 0.00028278 and the intercept is 44.54401034. Again, the majority of the values for the graph are located in zone 2.

Additionally, an average can be calculated to determine an optimal regression equation for use with Blender objects. For example, based on the three objects displayed in this section, the average slope can be calculated as 0.000158322742 and the intercept as 45.03072705. This linear regression can give an approximation for the second zone for any of the previous three objects to a certain accuracy.

Ultimately, the model generated is accurate for one object specifically, however with more models generated for more objects, an average value for both the slope and y-intercept can be determined. This final average can be used to estimate the GPU percentages per triangle for any object.

**Conclusion**

The EAS Lab utilized many techniques to analyze the relationship between GPU utilization and its effect on smartphone resources. In order to begin the analysis for GPU utilization, an initial database of 50 blender objects was created. Each object is unique, represents different real life objects, and contains a different triangle amount. The objects were saved as .obj files. The textures were applied to each object and saved as .mtl files. This process finalized the creation of the initial database.

Throughout the duration of the thesis, three main projects were completed to explore methods to predict and analyze GPU utilization. The projects required the use of tools such as Blender, Python, Java, and Excel. Additionally, the scripts to run blender required the use of the linux operating system. Using these tools, the projects were able to be completed in an efficient manner.

The first project involved creating a Java program that calculates the average GPU utilization percentage between each time interval for a specific blender object. The initial GPU utilization script was executed on linux and the date, time, GPU percentage, and triangle count were recorded into a notepad file. The values were imported into the Java program and were spliced and organized into different arrays. Once organized, the average percentage values were calculated for each time interval and displayed on screen. Following the execution of the algorithm, the results showed a direct correlation between the increase in the number of triangles and an increase in GPU percentage utilization. These results provided valuable insight to how triangle counts can affect GPU performance for a smartphone.

The second project involved creating a Python script that calculates the coefficient values for \( a \), \( b \), \( c \), and gamma for the degradation formula. The initial degradation formula is in a polynomial form that can be improved to make it easier to implement in a linear regression. The formula was rearranged to form a multivariate linear regression. Once modified, a script for blender was executed to produce degradation errors for multiple ratios. Multiple gamma values were calculated and used to create multiple different

The third project involved creating a Java program that calculates the average GPU utilization percentage between each time interval for a specific blender object. The initial GPU utilization script was executed on linux and the date, time, GPU percentage, and triangle count were recorded into a notepad file. The values were imported into the Java program and were spliced and organized into different arrays. Once organized, the average percentage values were calculated for each time interval and displayed on screen. Following the execution of the algorithm, the results showed a direct correlation between the increase in the number of triangles and an increase in GPU percentage utilization. These results provided valuable insight to how triangle counts can affect GPU performance for a smartphone.
models for the degradation formula. The model with the lowest RMSE value was selected to be the main model for degradation. This model was then used to estimate the coefficient values for ‘a’, ‘b’, and ‘c’. These results provided values for the variables needed to accurately predict degradation values for new distances or ratios.

The third project involved creating a Python script that calculates a model for GPU utilization. A normal GPU utilization trend can be divided into three ‘zones’. The first zone is the initial GPU without any processing of objects, the second zone is the phase in which the GPU utilization percentage is increasing based on an increase in triangles, and the third zone is when the GPU utilization has flattened out. The zone of importance is the second zone. The second zone can be isolated using the minimum and maximum values for the regression. Once isolated, the values are used to calculate a linear model. Using multiple models, an average can be calculated that can be used for many objects.

The ultimate goal of the EAS-Lab is to create more efficient energy performance and reliability. These projects focused on trying to explore methods to quantitatively analyze the relationship between the GPU utilization and the number of triangles. The focus was to create models to help predict GPU values for any ratio, triangle count, or distance for each object. Ultimately, with enough accuracy, these models can be used to predict those values for multiple objects instead of just one and help explore methods to increase efficiency.
References
