Performance Analysis of OpenGL Java Bindings with Graphics Scan Conversion Algorithms

Naresh Purohit1, Rajendra Purohit2
1Department of Computer Science Engineering, Mits - Jadan, Pali, Rajasthan, India
2Research Scholar, Rajasthan Technical University, Kota, Rajasthan, India

ABSTRACT

This paper presents a Performance Analysis of OpenGL with Java bindings i.e. Light Weight Java Game Library (LWJGL). Three different sections, i.e. Line Rendering, Circle Rendering, and Image Rendering have been selected as the benchmarks for extensive analysis of the performance gaps between the two. The results show the performance of LWJGL with rendering algorithms to display from a single pixel to some large images in the screen on the basis of execution time analysis. LWJGL is a Java-Binding of OpenGL that enables developing portable, interactive 2D and 3D graphics applications.

Keywords: OpenGL, LWJGL, Rendering, Pixel, Java Bindings.

I. INTRODUCTION

The Computer Graphics is one of the most effective and commonly used methods to communicate the processed information to the user. It displays the information in the form of graphics objects such as pictures, charts, graphs and diagram instead of simple text.

In computer graphics, pictures or graphics objects are presented as a collection of discrete picture elements called pixels. The pixel is the smallest addressable screen element.[4]

A. Rendering

Rendering is the process of generating an image from a model, by means of a software program. The model is a description of three dimensional objects in a strictly defined language or data structure. It would contain geometry, viewpoint, texture and lighting information.

B. LWJGL (Light Weight Java Game Library)

LWJGL is a Java binding for OpenGL API. The Lightweight Java Game Library (LWJGL) is an open-source Java software library for video game developers. It exposes high performance cross-platform libraries commonly used in developing video games and multimedia titles. It provides a way for Java developers to get access to resources that are otherwise unavailable or poorly implemented on the existing Java platform.[3]
II. RENDERING OBJECTS

In this paper, the rendering is done on linear and non-linear objects. In linear objects, straight line algorithms and for non-linear objects, circle algorithms are analysed.

A. Line Drawing

A line drawing algorithm is a graphical algorithm for approximating a line segment on discrete graphical media. On discrete media, such as pixel-based displays and printers, line drawing requires such an approximation (in nontrivial cases). Basic algorithms rasterizing lines in one color. A better representation with multiple color gradations requires an advanced process, spatial anti-aliasing. A line connects two points. It is a basic element in graphics. To draw a line, we need two points between which we can draw a line.

A.1 LSI Algorithm

The Cartesian slope-intercept equation for a straight line is with m representing the slope of the line and b as the y intercept. [5]Given that the two endpoints of a he segment are specified at positions (x1, y1) and (x2, y2), as shown in Fig. , we can determine values for the slope m and y intercept b with the following calculations:

Table 1. LSI Rendering through LWJGL

<table>
<thead>
<tr>
<th>Execution no.</th>
<th>Machine Execution Time</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>218.690</td>
<td>227.911</td>
</tr>
<tr>
<td>2</td>
<td>232.882</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>246.748</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>217.392</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>223.843</td>
<td></td>
</tr>
</tbody>
</table>

Step 1: Calculate
\[ dx = x2-x1 \]
\[ dy = y2-y1; \]
\[ m = dy/dx; \]
\[ b = y1-m*x1; \]

Step 2: check if if(m<=1)
Then
Calculate \( y = mx+b \)
For loop x1 to x2 times each iteration
Draw pixel(x,y) position
Else
Calculate \( x = (1/m)y+b; \)
For loop y1 to y2 times each iteration
Draw pixel(x,y) position
A.2 DDA Algorithm

The digital differential analyser (DDA) is a scan-conversion line algorithm based on calculating either dy or dx. Using dx or dy, we sample the line at unit intervals in one coordinate and determine corresponding integer values nearest the line path for the other coordinate.[6]

**Step 1:** Get the input of two end points \((X_0,Y_0)\) and \((X_1,Y_1)\).

**Step 2:** Calculate the difference between two end points.

\[\begin{align*}
\text{dx} &= X_1 - X_0 \\
\text{dy} &= Y_1 - Y_0
\end{align*}\]

**Step 3:** Based on the calculated difference in step-2, you need to identify the number of steps to put pixel. If \(dx > dy\), then you need more steps in \(x\) coordinate; otherwise in \(y\) coordinate.

\[
\text{if (absolute(dx) > absolute(dy))}
\]

Steps = absolute(dx);

\[
\text{else}
\]

Steps = absolute(dy);

**Step 4:** Calculate the increment in \(x\) coordinate and \(y\) coordinate.

\[
\begin{align*}
\text{Xincrement} &= \text{dx} / (\text{float}) \text{steps}; \\
\text{Yincrement} &= \text{dy} / (\text{float}) \text{steps};
\end{align*}
\]

**Step 5:** Put the pixel by successfully incrementing \(x\) and \(y\) coordinates accordingly and complete the drawing of the line.

\[
\text{for(int v=0; v < Steps; v++)}
\]

\[
\begin{align*}
x &= x + \text{Xincrement}; \\
y &= y + \text{Yincrement};
\end{align*}
\]

\[
\text{putpixel(Round(x), Round(y));}
\]

<table>
<thead>
<tr>
<th>Execution no.</th>
<th>Machine Execution Time</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>210.320</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>222.683</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>227.246</td>
<td>221.631</td>
</tr>
<tr>
<td>4</td>
<td>219.858</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>228.048</td>
<td></td>
</tr>
</tbody>
</table>

**Table 2. DDA Rendering through LWJGL**

**A.3 Bresenham's Line Algorithm**

The Bresenham algorithm is another incremental scan conversion algorithm. The big advantage of this algorithm is that it uses only integer calculations. Moving across the \(x\) axis in unit intervals and at each step choose between two different \(y\) coordinates.[1]

**Step 1:** Input the two end-points of line, storing the left end-point in \((x_0,y_0)\).

**Step 2:** Plot the point \((x_0,y_0)\).

**Step 3:** Calculate the constants \(dx, dy, 2dy,\) and \((2dy - 2dx)\) and get the first value for the decision parameter as 

\[
p_0 = 2dy - dx
\]

**Step 4:** At each \(X_k\) along the line, starting at \(k = 0\), perform the following test –

If \(p_k < 0\), the next point to plot is \((x_k+1,y_k)\)

\[
p_{k+1} = p_k + 2dy
\]

Otherwise,
pk+1 = pk + 2dy – 2dx

**Step 5:** Repeat step 4 (dx – 1) times.
For m > 1, find out whether you need to increment x while incrementing y each time.

**Table 3.** Bresenham’s Line Rendering through LWJGL

<table>
<thead>
<tr>
<th>Execution no.</th>
<th>Machine Execution Time</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100.419</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>100.257</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>103.189</td>
<td>106.6434</td>
</tr>
<tr>
<td>4</td>
<td>119.704</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>109.648</td>
<td></td>
</tr>
</tbody>
</table>

Time in Milliseconds

**Figure 4.** Basic Circle Equation Drawing

for(x = -r; x <= r; x++)
y = SquareRoot of (r * r - (x * x));
Draw Pixel(xc+x,yc+y) for the upper half
DrawPixel(xc+x,yc-y) for the lower half

**Table 4.** Basic Circle Equation Rendering through LWJGL

<table>
<thead>
<tr>
<th>Execution no.</th>
<th>Machine Execution Time</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>163.138</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>163.166</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>171.513</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>194.679</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>159.192</td>
<td></td>
</tr>
</tbody>
</table>

Time in Milliseconds

**Chart 3.** Bresenham’s Line Rendering through LWJGL

**B. Circle Drawing**

**B.1 Circle Equation Drawing**

The circle is a frequently used component in pictures and graphs, a procedure for generating either full circles or circular arcs is included in most graphics packages. More generally, a single procedure can be provided to display either circular or elliptical curves.

A circle is defined as the set of points that are all at a given distance R from a center position (Xc, Yc).

This distance relationship is expressed by the Pythagorean Theorem in Cartesian coordinates as

\[(X-Xc)^2 + (Y-Yc)^2 = R^2\]

**Chart 4.** Basic Circle Equation Rendering through LWJGL

**B.1 Bresenham’s Circle Drawing**

The purpose of Bresenham’s circle algorithm is to generate the set of points that approximate a circle on a pixel-based display. We generate the points in the first octant of the circle and then use symmetry to generate the other seven octants. We start at (r, 0). This pixel is chosen trivially. Next we need to find
the pixel for the $y = 1$ row. We must make a decision between two candidate points.[2]

![Figure 5. Eight Way Symmetry](image)

**Figure 5.** Eight Way Symmetry

**Step 1:** Set $X = 0$ and $Y = R$
Set $D = 3 - 2R$

**Step 2:** Repeat While ($X < Y$)
Call Draw Circle($Xc, Yc, X, Y$)
Set $X = X + 1$
If ($D < 0$) Then
$D = D + 4X + 6$
Else
Set $Y = Y - 1$
$D = D + 4(X - Y) + 10$ [End of If]
Call Draw Circle($Xc, Yc, X, Y$) [End of While]
Exit

Draw Circle ($Xc, Yc, X, Y$):
Call PutPixel($Xc + X, Yc, + Y$)
Call PutPixel($Xc - X, Yc, + Y$)
Call PutPixel($Xc + X, Yc, - Y$)
Call PutPixel($Xc - X, Yc, - Y$)
Call PutPixel($Xc + Y, Yc, + X$)
Call PutPixel($Xc - Y, Yc, + X$)
Call PutPixel($Xc + Y, Yc, - X$)
Call PutPixel($Xc - Y, Yc, - X$)
Exit

**Table 5.** Bresenham’s Circle Rendering through LWJGL

<table>
<thead>
<tr>
<th>Execution no.</th>
<th>Machine Execution Time</th>
<th>Average Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>77.239</td>
<td>88.1312</td>
</tr>
<tr>
<td>2</td>
<td>42.876</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>98.883</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>122.418</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>99.240</td>
<td></td>
</tr>
</tbody>
</table>

**III. CONCLUSION**

In this paper, there are two section of different performance analysis on OpenGL. Three algorithms are used to render lines, two algorithms are used to render circles pixel by pixels. In our test and analysis, we measures actual execution time for rendering objects on the screen by Light Weight Java Game Library. By this analysis we can develop large level graphics applications like high definition games and many more. Thus JWJGL, a java programming binding for OpenGL seems to be a better choice for developing games and other graphics applications where achieving as high performance as possible is the main priority.

**IV. FUTURE WORK**

Computer graphics is used today in many different areas of industry, business, government, education, entertainment, and most recently, the home. The list of applications is enormous and is growing rapidly as computers with graphics Capabilities become commodity products. OpenGL is presently used to produce both accurate and schematic representations of geographical and other natural phenomena from measurement data. Examples include geographic maps, relief maps, exploration maps for drilling and mining, oceanographic charts, weather maps, contour maps, and population-density maps. These algorithms for object drawings probably to create 2D and 3D graphs of mathematical, physical, and
economic functions; histograms, bar and pie charts, task-scheduling charts, inventory and production charts.

V. REFERENCES

[1]. Development of the Bresenham’s Line Algorithm, Alfred L. McKinney, K.K. Agarwal, Department of Computer Science, Louisiana State University in Shreveport, Shreveport, LA 71115

[2]. A Fast Bresenham Type Algorithm For Drawing Circles by John Kennedy Mathematics Department Santa Monica College 1900 Pico Blvd. Santa Monica, CA 90405.


[5]. Improved Line Drawing Algorithm: An Approach and Proposal, Muhammad Usman Khan & Prof (Dr) Md. Rizwan Beg, Integral University Lucknow, India.

[6]. Prof. Sukhendu Das Dept. of Computer Science and Engineering Indian Institute of Technology, Madras Lecture - 13 Scan Converting Lines, Circles and Ellipses.


[8]. Learning Java Bindings for OpenGL (JOGL) by Gene Davis

[9]. Volume Rendering Using OpenGL and Extensions, M. Meissner; U. Hoffmann; W. Strasser, IEEE, 1070-2385