**Course Code: CST-471, IT -471** **Couse Name: Computer Graphics Lab**

**Learning Objectives:**

1. To learn the principles and commonly used paradigms and techniques of computer graphics. To provide students with a foundation in graphics applications programming.
2. To gain a proficiency with OpenGL, "a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics."
3. To develop a facility with the relevant mathematics of computer graphics and understand 3D visualization.
4. To understand, visualize and control the real time universe.
5. The massive data sets being produced by cheap sensors are useless unless they can be understood by people. Complicated machines are useless unless they can be easily controlled. This course will help you do both. The key is graphics and visualization. We don't just teach useful platform-independent tools. We also teach the underlying math and algorithms used by all tools so that you can design better tools.

**Official Lab Course Description:**
This lab course would cover practical assignments broadly to address: interactive computer graphics; 2-D and 3-D rasterization and rendering pipelines, including geometric object and view transformations, projections, hidden surface removal; lighting models for local and global illumination; free form drawing with Curves; hierarchical modeling of 3-D objects; systems and libraries supporting display and user interaction.

**Topics and Goals:**
This course deals with the fundamentals of computer graphics. We will emphasize the most basic algorithms and concepts in computer graphics that form the foundation (both historical and practical) for most modern graphics systems. The emphasis in this class will be on interactive 3D computer graphics, although we will discuss some 2D algorithms, and introduce some non-interactive rendering methods.

***Goals:***

- Become familiar with basic 2D rendering concepts and algorithms such as line drawing, line and polygon clipping, polygon filling, and antialiasing.
- Understand 3D rendering techniques including hierarchical model structures, geometric transformations, projections, and hidden surface removal.
- Understand basic lighting and shading techniques.
- Understand the basics of color models as they relate to computer graphics.
- Become familiar with basic aspects of geometric and solid modeling and curves.
- Be able to write basic graphics programs using OpenGL.

***Topics:***

1. Overview of graphics – applications, history, technologies.
2. 2D primitive drawing, including scan conversion, polygon filling, clipping, and antialiasing.
3. 3D geometric transformations including translation, scaling, rotation.

4. Specification of view frusta and projections (parallel and perspective).
5. Hidden-surface removal and z-buffering.
6. Color models and blending (transparency)
7. Basic illumination (including flat, Gouraud, Phong) and shading.
8. Texture mapping.
9. Hierarchical 3D model specification.
10. Basics of geometric and solid modeling (e.g. curves and surfaces).
11. Ray-tracing basics.
12. Event-driven programming.
13. Basics of programmable shades.
14. OpenGL syntax and programming.

If time permits, we may be able to cover certain other topics like fractals. This would be announced separately during the session.

Upon finishing this class, you should be able to understand how graphics are produced, write your own basic graphics programs, and explore on your own some of the more advanced graphics techniques and algorithms

**Prerequisites:**

1. CST203 Data Structures and Algorithms.

2. CST306 Object Oriented Analysis and Design (Programming in C/C++).

3. Linear Algebra.

Though these are the only formal prerequisites for this course, this class is an upper-level computer science course. As such, there will be a significant amount of programming in this class, and students will be expected to use C/C++ for the assignments. If you are not comfortable programming in C/C++, you should not take this class.

In addition, students should be familiar with *basic* linear algebra (vector and matrix representation and arithmetic). A prior course in linear algebra is more than sufficient, and many students may have had enough previous exposure without such a class. A review of most of the relevant material is given in the textbook's Appendices.

**Required Textbooks:**

**1.** Hearn, Baker, Carithers. *Computer Graphics with Open GL.* Fourth Edition, Pearson Prentice Hall, 2010.

The Hearn and Baker book will be the primary textbook for all required readings. It gives a broad overview of graphics fundamentals, as well as a broad description of OpenGL.

**2.** The OpenGL Red Book:

Shreiner. *Open GL Programming Guide, Seventh Edition*. Addison-Wesley, 2009.
OR: a (free) online earlier version of one of the earliest editions of the book is available at: http://www.opengl.org/documentation/red_book/

Although Hearn and Baker include OpenGL information, the Programming Guide is far more complete, better organized, etc. The OpenGL Programming guide is probably the best overall guide to OpenGL programming. OpenGL has become a much more dynamic library, and new versions of the OpenGL standard are regularly released. With many of these come new editions of the Programming Guide. The 7th edition is the most recent one available, and covers OpenGL through version 3.1. An 8th edition, covering OpenGL 4.1 will come out soon. An online version of an early edition (probably first or second) is available at the link above; while very outdated, it still has several fundamental ideas included.

For the purposes of this class, you will be able to get by with the material in the main textbook, and the online version. However, if you plan to work with graphics further, especially in OpenGL, you would probably find it worthwhile to purchase a more recent copy of the Red Book.

**Lab Grade Distribution**

There will be several (6-8) homework assignments throughout the semester covering the 30% of the course grade point average. These may involve written work, programming, or both, and may vary significantly in difficulty.
The expected grading scale will be AA 90% > AB 80% > BB 70% > BC > CC> 60% > D>F. Depending on the final percentage distribution, an absolute or relative curve may be applied, though an attempt will be made to avoid this. In addition, the instructor reserves the right to raise grades near a "borderline" to the next highest letter grade. Factors weighing into this decision will be the individual student's perceived effort, class attendance and participation and submission of assignments on time.

5% class participation
10% Assignment 1 - Primitives
10% Assignment 2 – Clipping and Filling
15% Assignment 3 – Transformations
15% Assignment 4 - Projections
15% Assignment 5 – Hidden Surface Removal
15% Assignment 6 – Light Illumination and Shading
15% Assignment 7 - Curves

**Policies:**

*Attendance:* Attendance will not be checked; however it will be a consideration in decisions about whether a final grade near a borderline might be raised. Students with unexcused absences should not expect additional help outside of class and are still responsible for any material or instructions given in class, for turning in assignments on time, and for taking exams at the scheduled times. Make-up assignments or exams will not be given unless there is a highly unusual circumstance. If you know that you will be absent for a class, you should let the instructor know ahead of time.

*Turning in Assignments:* The due date/time for assignments will be given with each assignment. Generally, the due time will be the last hour on the due date, and assignments will need to be submitted on time. The due time would be usually the third lab of every assignment announced in class.

*Late Assignments:* The total number of weeks, w, that assignments are late will be determined. The score on the assignment will be multiplied by a late penalty of $0.9998^{w}$. For example, if an assignment is 1 week late, you lose a little over 1% of the grade. If it is one week late, you lose about 25%. After 2 weeks, you will have lost 44%, and after 3 weeks you will have lost 58%. For some assignments, there might be no late assignments accepted after a certain point; this will be stated when applicable.

*Communication:* Communication outside of class will be through two mechanisms, which students should be made aware of:

a) Email will be sent. Students are expected to read all such email, and should ensure that their email address is correct.
b) Through notice boards. A class web page (listed at the top of this syllabus) will be maintained throughout the semester; assignments, schedule, and other notes will be posted here. Students are responsible for checking the notice boards/ web page regularly for class updates.

*Reading:* Readings from the required textbook will be given out throughout the semester. We will not be able to discuss all of it in class, but you are still responsible for keeping up with these readings.

*Code Documentation:* When assignments are graded, source code may be examined to verify the way a solution was achieved or to award partial credit. It is *your* responsibility to make sure that your source code is presented in a clear, readable, way. Even if your code "works," if the grader can't understand it, you may lose points.

*Academic Honesty*: A student should not lie, cheat, or steal or tolerate those who do.To accept responsibility for learning, and to follow the philosophy and rules of the system. Students will be required to state their commitment on examinations, lab assignments, take home assignments, and other academic work. Ignorance of the rules does not exclude any member the requirements or the processes of the system.

For this class, the interpretation of the code will be as follows: Unless specifically stated otherwise, all assignments are to be done **on your own**, unless specified otherwise. You may discuss general concepts, and get help in tracking down a persistent bug, but should not copy work, download code from the web or other sources, or work together with other students on problems or programs unless specifically stated otherwise. By turning in an assignment or exam, you are implicitly assumed to be committing to the academic honesty. **If you are unsure of whether a type of cooperation is appropriate, check with the course coordinator or instructor or TA first.** That is, you should err on the side of assuming cooperation is *not* allowed.

**List of Assignments**

1. **Primitives:** The purpose of this programming assignment is to introduce students to graphics programming with OpenGL and to increase the understanding of the Bresenham technique for drawing a line, circle and an ellipse. You have to include the path of the OpenGL library in the C/C++ program then initialize graphics.

Viewing Transformation: Define a Window(Xwmin, Ywmin, Xwmax, Ywmax) in user coordinates and map it to a Viewport(Xvmin, Yvmin, Xvmax, Yvmax) in device coordinates. Construct multiple windows and map them to different viewports. Map atleast one of the windows in negative coordinates to a viewport. Now in this mapped viewport draw various inbuilt primitives and practice with their various attributes.

All the rest of the assignments would be using this mapping for the display. None of the assignment would be accepted in device coordinates.

Scan Conversion: You are to write procedures as drawLine.c, drawCircle.c, drawEllipse.c and drawPolygon.c which are used to generate line, circle, ellipse and polygons respectively. Use the OpenGL routine to draw a point, and using Bresenham techniques discussed in class and in your textbook draw lines, circles, and ellipse. For drawing a polygon you could use your already defined drawLine.c to draw the polygon boundary.

The aim is to i) To study and Implement Midpoint line algorithm given the two end points of the line. The algorithm should handle all slopes and be able to draw line in all the four quadrants and with any limit in any order of the end points. Ii) To study and implement Midpoint circle algorithm given the points of the center and the radius. You have to design a program which will take input the coordinates of the center of the circle as well as the radius. ii) To study and Implement Midpoint Ellipse algorithm. You have to design a program which will take input the coordinates of the center of the ellipse as well as the major and minor axis. Like,
*Enter the coordinates of the centre:*

*X-coordinate = 350*

*Y-coordinate = 250*

*Enter the Minor axis: 50*

*Enter the Major axis: 100*

After that you have to draw the ellipse with the supplied parameters.

I should be able to compile and execute your final product on a Linux system.

Drawing Polygons: Using the above line primitive draw a polygon (having atleast three planar surfaces). The polygon is defined as an object containing multiple surfaces, where a surface is defined as a plane Ax + By + Cz + D = 0 and has a property color.
Now draw the polygon as a list of multiple planar surfaces. Where every planar surface is defined as a list of edges and every edge is defined as a pair of vertices. From the vertex list of a polygon surface you could calculate the planar coefficients defined above.

**Assessment**

Your overall programming assignment grade will be based on the following: Does your *drawLine* routine handle all kinds of lines (shallow or steep/positive or negative slopes, vertical, diagonal, or horizontal lines), starting from either endpoint? Does your drawCircle and drawEllipse routine handles circle (ellipse) with any point as origin? Does your *drawPolygon* routine generate both concave and convex polygons; regardless of order of entry of vertices.

**2. Filling and Clipping:** Filling: To study and Implement Polygon Filling: Implement polygon filling algorithms discussed in class for a single surface a) using Boundary Filling Algorithm (4 and 8 connected) and b) Scan line polygon filling approach.

Your program should accept a polygon as a list of vertices in counterclockwise direction and then fill it with a fill color using the above techniques taught in class. The program must be able to handle convex polygons.

Clipping: To study and Implement Line Clipping with Cohen- Sutherland Algorithm. Implement the Liang-Barsky Algorithm and Parametric clipping algorithm in the same manner for clipping a line. Study and implement polygon clipping algorithm using Sutherland – Hodgeman algorithm.

You have to design a program which will take input the coordinates of two end points of a line and two window coordinates. Then you have to draw the line and clipped window.

Like,

Enter Minimum window co-ordinates:- 200 250

Enter Maximum window co-ordinates:- 300 350

Enter co-ordinates of first point of line: - 180 250

Enter co-ordinates of second point of line: - 200 300

The Cohen-Sutherland algorithm uses a divide-and-conquer strategy. The line segment's endpoints are tested to see if the line can be trivially accepted or rejected. If the line cannot be trivially accepted or rejected, an intersection of the line with a window edge is determined and the trivial reject/accept test is repeated. This process is continued until the line is accepted.

To perform the trivial acceptance and rejection tests, we extend the edges of the window to divide the plane of the window into the nine regions. Each end point of the line segment is then assigned the code of the region in which it lies.

**Assessment**

Your overall programming assignment grade will be based on the following: Does your *fillPolygon* routine fill rectangles and triangles and both concave and convex polygons properly; regardless of order of entry of vertices.

Does your clipping algorithm perform trivial acceptance and rejection tests? You should be able to explain the efficiency of the clipping algorithms and also be able to explain why Sutherland-Hodgeman algorithm works only for convex clipping regions?

**3. Transformations:** To implement a set of basic transformations on an (object) Polygon i.e. Translation, Rotation and Scaling. To study and implement a set of composite transformations on polygons i.e. reflection, shear (x &y), and rotation about an arbitrary point and line**.** Aim is to study and implement Mouse and Keyboard interaction with OpenGL programs; To study events in OpenGL; How to handle them and use them inside an OpenGL Program.

Use your experimentation with the object or Polygon defined in Assignment 1. Apply all transformations studied in class and as explained in your textbook to answer the following questions: (BE SURE to reset the parameters between each question to isolate the cause and effects. All the transformations should be key driven. For example for translation in +ve x direction press ‘t’, and for –ve x direction ‘Shift t’, etc.) This can be done using the key function menu that appears if the right mouse button is held down while over the textual area or by typing "m".)

Please use graphics terminology such as translation, scale, and rotation, and if rotation, about which axis and whether clockwise or counterclockwise. If the polygon did not change as you expect, hypothesizing why would be helpful to you. For example, Rotation about an arbitrary point:

This is done by three transformation steps: translation of the arbitrary point (Xc,Yc) to the origin, rotate about the origin, and then translate the center of rotation back to where it belongs. To transform a point, we would multiply all the transformation matrices together to form an overall transformation matrix.

**Assessment**

After transformations assignments you should be able to answer:

How does the image change if you change the *x/y/z* argument of the translate parameter list to negative? to positive?

How does the image change if you change the *angle* argument of the rotation parameter list to negative? to positive? (rotation about the X/Y/Z axis)

How does the image change if you are transforming (translate/rotate/scale/reflect/shear) an object about an arbitrary point or line?

How does the image change if you make the *x/y/z* scale factor smaller? larger?

Rotate the object by 360 degrees so that you can see the entire side of the object.

**4. Projections:** Aim is to understand 3D viewing by defining different planes of projections and different center of projections. To understand setting of a a viewing direction and the viewing pipeline.

Use your experimentation on a Unit Cube defined as a polygon with six planar surfaces in different colors. You should experiment with the complete classification of planar geometric projections. (This assignment is also to be implemented as key driven. For example, for orthographic projection press 'o' key, for isometric 'i', for diametric 'd', trimetric 't' etc).

The aim is to experiment with the complete classification of projection (Parallel: Orthographic (and Axonometric: isometric, diametric, trimetric), Cavalier, Cabinet, General and Perspective: one point, two point and three point perspective) studied in the class and your textbook to answer the following questions.

**Assessment:** You should be able to answer the following questions.

How does the image change if you make the plane of projection as the principal planes and the center of projection is at infinity in the direction perpendicular to the plane of projection?

How does the image change if you define the plane of projection with a reference point and normal to the plane; and the center of projection is at infinity in the direction perpendicular to the plane of projection?

How does the image change if you define the plane of projection with a reference point and normal to the plane; and the center of projection is at a point i) on any of the axis, ii) or it is in a plane or iii) it is in space.

What is the main difference between parallel and perspective projection?

What is the main difference between orthographic and axonometric projections (isometric, diametric and trimetric projections)?

Show the difference between isometric, diametric and trimetric projections.

What is the difference between cavalier, cabinet and general parallel projections?

What is the difference between one point perspective, two point perspective and three point perspective projections?

**5. Hidden Surface removal:** Aim is to understand the need of hidden surface removal or realistic viewing.
Implement back face removal technique on a unit cube (every surface should have a different color). Show the change in visible surfaces by changing the viewing position.
Implement two of the polygon visible surface algorithms studied in the class such as Z- Buffer algorithm, scan line algorithm, painter's algorithm, BSP tree algorithm or area subdivision algorithm.
Implement a simple ray tracer for spheres and polygons. Use OpenGL features for showing the rendering effect.

**Assessment:**
You should be able to answer the following questions.
What is object space and image space?
How many numbers of surfaces Z- buffer algorithm can handle?
Do we need to sort (or input in order) surfaces for a Z-buffer algorithm?
Can we handle concave surfaces with back face removal algorithm?
Which hidden surface removal algorithm should be preferred if the number of surfaces is large?
Which hidden surface removal algorithm should be preferred if the number of overlapping surfaces is large?
How can the algorithms discussed be adapted to work with polygons containing holes?
How might you modify the BSP-tree algorithm to accept objects other than polygons?

**6. Light Illumination and Shading**. Aim is to understand how real-world lighting conditions are approximated by OpenGL. To understand how to render objects by defining light source, material, and lighting model properties. Implement a test bed for experimenting with local illumination models. Store an image that contains for each pixel its visible surface's index into a table of material properties, the surface normal, the distance from the viewer, and the distance from the normalized vector to one or more light sources. Allow the user to modify the illumination equation, the intensity and color of the lights, and the surface properties. Each time

a change is made, render the surface. Show the effect of Diffuse, Specular and Ambient illumination. Your renderer should support Constant, Gouraud and Phong shading models also.

**Assessment:** You should be able to answer the following.
What is rendering pipeline?
The difference between diffuse and ambient light sources.
Effect of material properties on the color of the surface.
Which surface materials produce good specular highlights?
How to approximate surface curvature to avoid mac-band effects?
How to compute surface normal?
How to interpolate shading using Phong and Gouraud shading models?

**7. Curves:** Aim is to understand free hand drawing. To understand nth order polynomial fitting techniques. To study parametric and non-parametric curves.

Write an interactive program to accept an arbitrary geometric matrix, basis matrix, and list of control points, and to draw the corresponding curves. Show the control polygon in a different color using a dashed line. Show the corresponding curve in another color using solid line. Show the control points. Find the conditions under which two joined Hermite curves have $C^0$ and $C^1$ continuity. Impose $C^0$, $C^1$ and $C^2$ continuity constraints on two Bezier curves. The two curves should be shown in two different colors. The continuity constraints should be imposed interactively with the corresponding effect should immediately. Analyze the effect of repeated control points and collinear points.
Expand the program that allows the user to interactively create and refine piecewise continuous cubic curves. Represent the curves internally as B-Splines. Allow the user to specify how the curve is to be interactively manipulated as – Hermite, Bezier or B-Splines.

**Assessment:**
You should be able to answer the following questions.
The relationship between number of control points and the degree of the curve.
The convex hull or the control polygon and the oscillations of the curve.
How to define a curve with boundary constraints, with basis matrix and with blending functions?
How to draw closed curves?
What is the effect of repeated control points?
How to impose continuity constraints to join two curves?
How to draw a circle using Bezier curves?