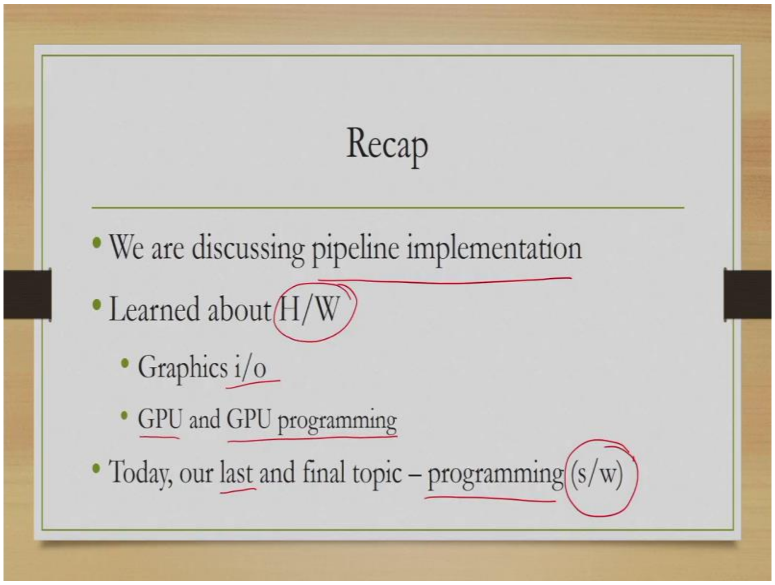
Computer Graphics
Professor Dr. Samit Bhattacharya
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati
Lecture 31

Hello and welcome to lecture number 31 in the course Computer Graphics. So, this is going to be our final lecture on the topic. So, far what we have discussed?

(Refer Slide Time: 00:44)



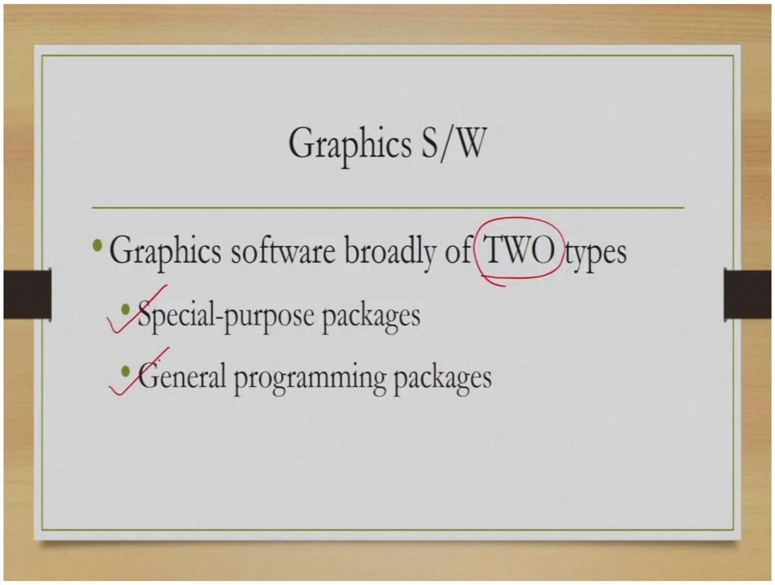
Recap

- We are discussing pipeline implementation
- Learned about H/W
    - Graphics i/o
    - GPU and GPU programming
- Today, our last and final topic – programming (s/w)

We discussed pipeline and then currently we are discussing pipeline implementation. In our earlier lectures we learned about the basic graphics hardware including the graphics input and output devices, the GPU or Graphics Processing Unit and the GPU programming basic idea. Today in our, this last topic we are going to learn about programming or how to write graphics programs that is essentially the software aspect of Computer Graphics.

Now, before we learn to program, we will first start with a basic introduction to graphics software. If you may recollect during our introductory lectures, we had a preliminary introduction but today we are going to recap as well as expand those discussions.

(Refer Slide Time: 01:55)

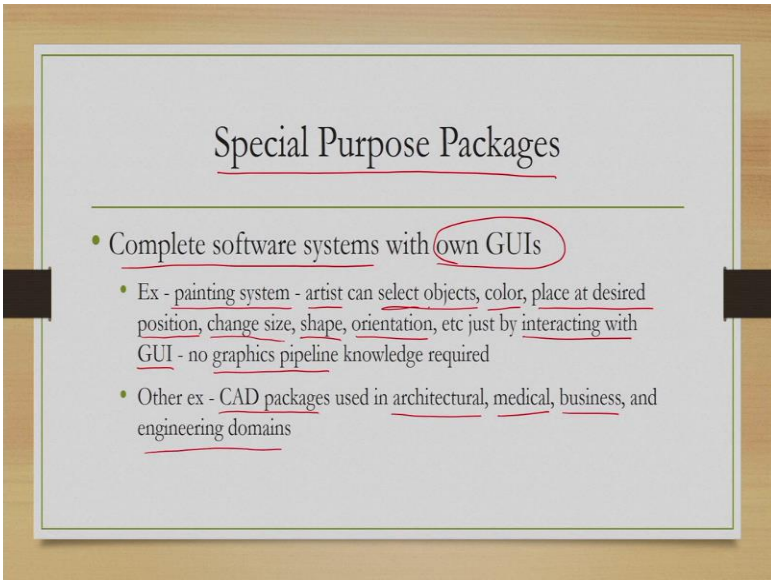


Graphics S/W

- Graphics software broadly of TWO types
  - Special-purpose packages
  - General programming packages

As we have mentioned earlier graphic software are broadly of two types. One is the special purpose packages and the other one is general programming packages.

(Refer Slide Time: 02:13)



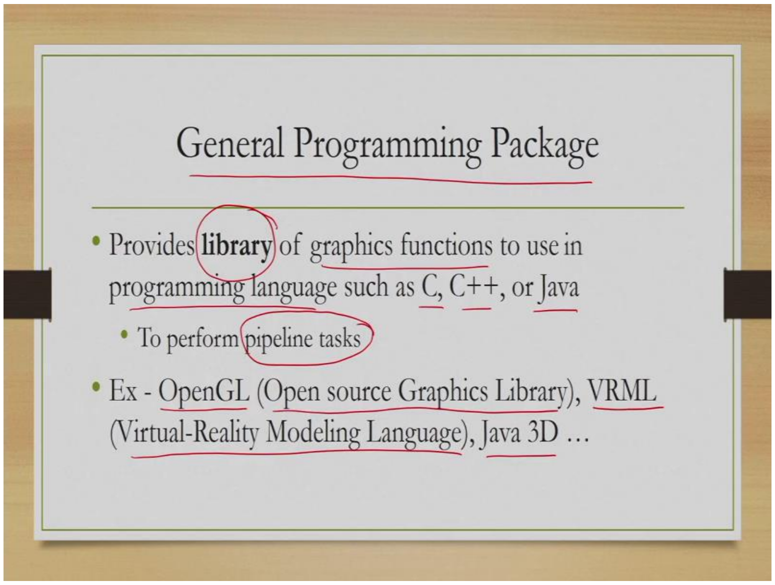
Special Purpose Packages

- Complete software systems with own GUIs
  - Ex - painting system - artist can select objects, color, place at desired position, change size, shape, orientation, etc just by interacting with GUI - no graphics pipeline knowledge required
  - Other ex - CAD packages used in architectural, medical, business, and engineering domains

In the special purpose packages, what we have? These are essentially complete software systems with their own GUIs or user interfaces. For example, painting system here it has its own user interface through which an artist can select objects, select colour, place the objects at desired position on the Canvas or the screen, change the size of the object, change the shape, also orientation and so on.

And all this, the artist can do by interacting with the user interface. So, there the artist need not know anything about the graphics pipeline or how it is implemented. These are examples of complete software systems or packages. Another example is the CAD package that we have learned about in the introductory lectures, CAD or Computer Aided Design packages. These are primarily used in architecture, medicine, business, engineering and such domains.

(Refer Slide Time: 03:46)



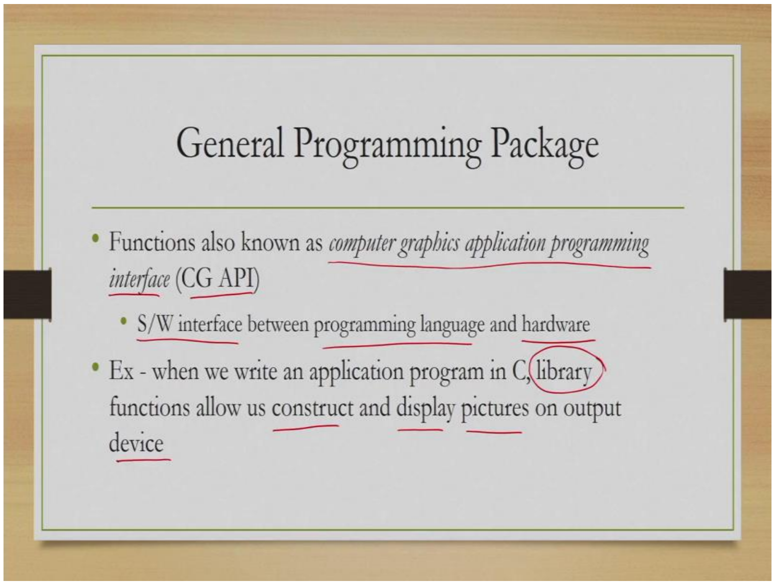
General Programming Package

- Provides **library** of graphics functions to use in programming language such as C, C++, or Java
  - To perform pipeline tasks
- Ex - OpenGL (Open source Graphics Library), VRML (Virtual-Reality Modeling Language), Java 3D …

The other type of software is the general programming package. Now, here we have libraries, libraries of graphics functions that are provided and we can use those libraries with any programming language such as C, C++, or Java and these functions are mean to perform or rather mean to help a programmer perform pipeline tasks. So, in other words they help the program and implement the pipeline.

An example is OpenGL, which stands for Open Source Graphics Library. Also there are VRML Virtual Reality Modeling Language, Java 3D and so on. So, there are many such libraries provided to implement graphics functions.

(Refer Slide Time: 04:55)

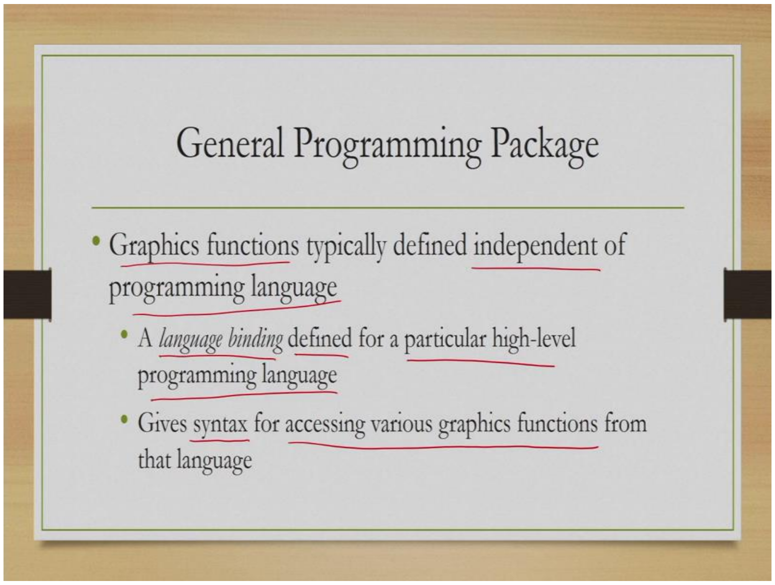


General Programming Package

- Functions also known as *computer graphics application programming interface* (CG API)
  - S/W interface between programming language and hardware
- Ex - when we write an application program in C, library functions allow us construct and display pictures on output device

Now, these functions are also known as computer graphics application programming interface or CG API. Now, they are essentially a software interface between programming language and the hardware. For example, when we want to write it an application program in a language say C, these library functions allow us to construct and display pictures on the output device. So, without these functions will not be able to do so.

(Refer Slide Time: 05:39)



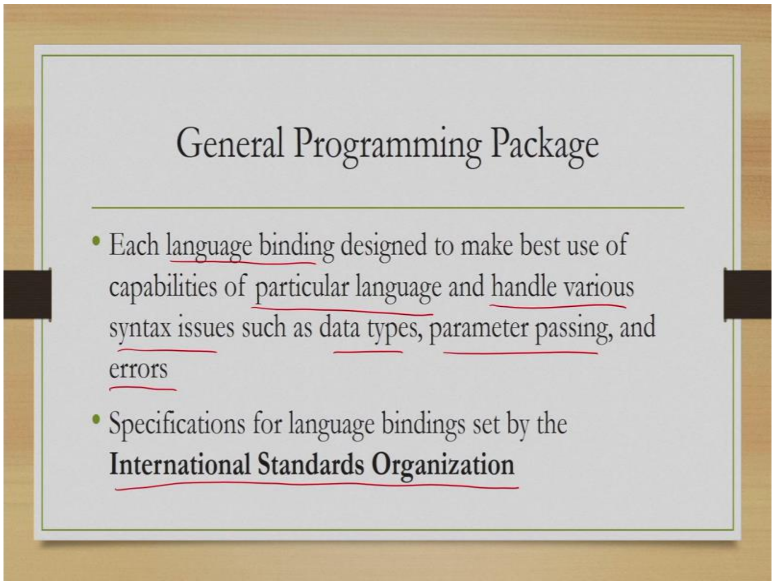
General Programming Package

- Graphics functions typically defined independent of programming language
  - A *language binding* defined for a particular high-level programming language
  - Gives syntax for accessing various graphics functions from that language

But one thing we should keep in mind is that the graphics functions are typically defined independent of the programming language and that is achieved through a concept called language binding. So, language binding is defined for a particular high-level programming language.

Now, through such binding we get the particular syntax to be used for accessing various graphic functions from that language. So, essentially language binding allows us to use these library functions from inside a program written using a particular language.

(Refer Slide Time: 06:33)



General Programming Package

- Each language binding designed to make best use of capabilities of particular language and handle various syntax issues such as data types, parameter passing, and errors
- Specifications for language bindings set by the **International Standards Organization**

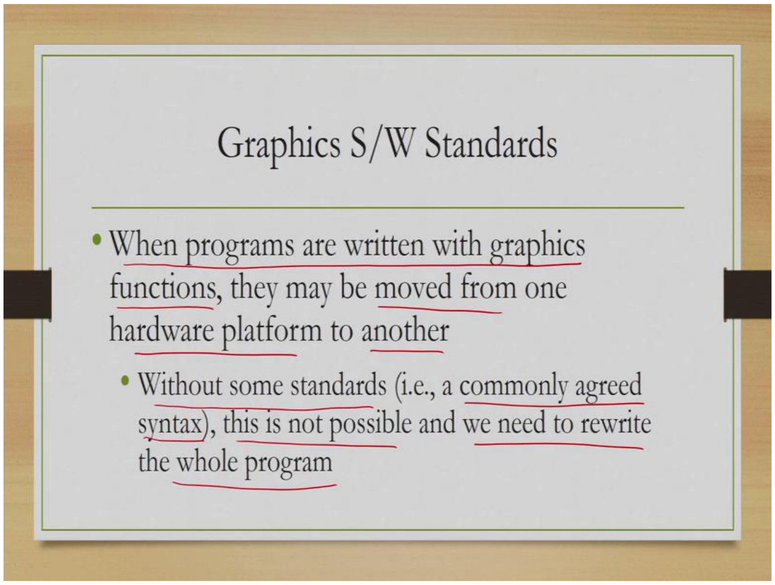
Now, each language binding is designed to make the best use of the capabilities there for a particular language and they are designed to handle various syntax issues such as data types, parameter passing and error handling. Now, these specifications or language binding are set by the ISO or International Standard Organization, so we need to know about these standards. We will have a brief introduction to different standards used for computer graphics.

(Refer Slide Time: 07:28)

S/W Standards

So, what are those standards, software standards that are used in computer graphics?

(Refer Slide Time: 07:32)

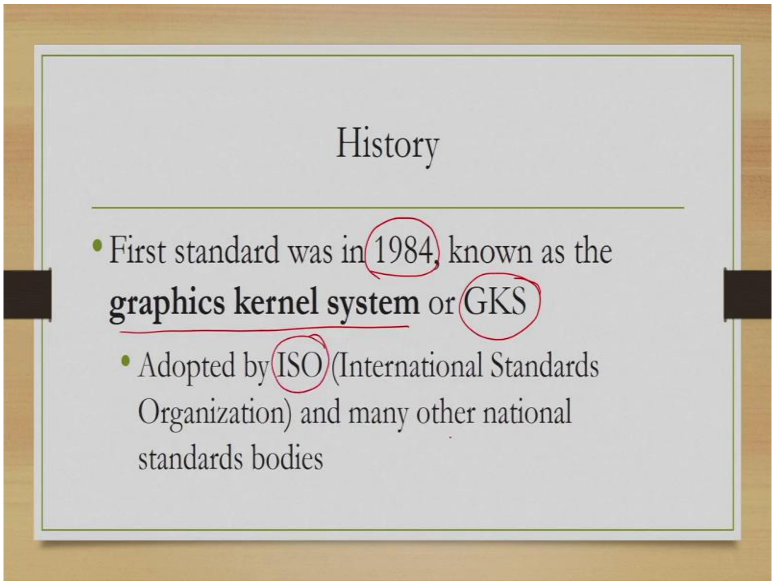


Graphics S/W Standards

- When programs are written with graphics functions, they may be moved from one hardware platform to another
  - Without some standards (i.e., a commonly agreed syntax), this is not possible and we need to rewrite the whole program

Now, why we need standard let us try to understand again. When we are writing a program with graphic functions, it may be the case that those programs are moved from one hardware platform to another. Now, how the computer will then understand the program if the platform is changed? There we require standard, without some standards which is essentially a commonly agreed syntax, this movement between platforms will not be possible and we need to rewrite the whole program. So, essentially we need to start from scratch. So, standard helps us avoid in such situation.

(Refer Slide Time: 08:33)

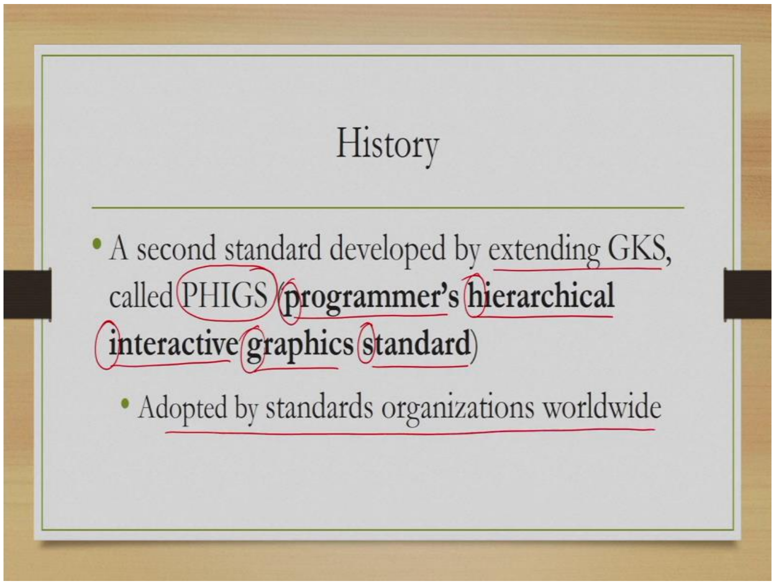


History

- First standard was in 1984, known as the **graphics kernel system** or GKS
  - Adopted by ISO (International Standards Organization) and many other national standards bodies

First graphic standard came in 1984, long ago which was known as the graphics kernel system or in short GKS. It was adopted by ISO as well as many other national standard bodies.

(Refer Slide Time: 08:57)

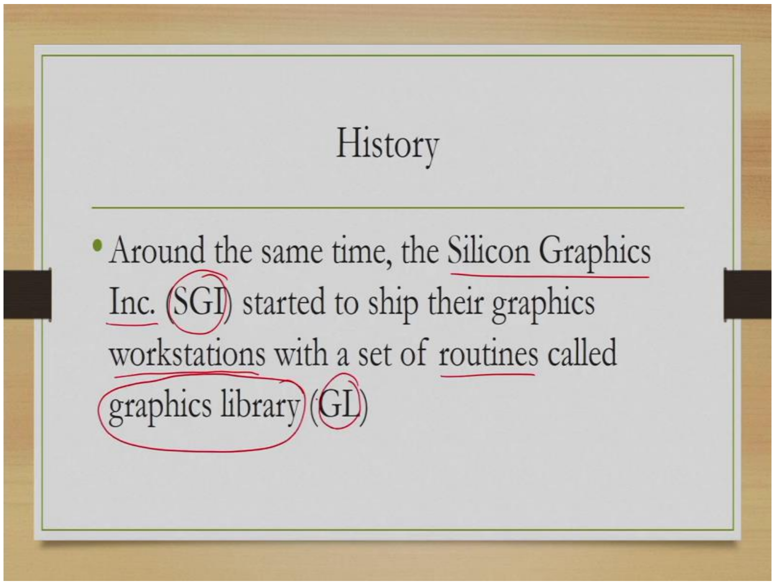


History

- A second standard developed by extending GKS, called PHIGS (**programmer's hierarchical interactive graphics standard**)
  - Adopted by standards organizations worldwide

Then came a second standard which was developed by extending the GKS, it was called PHIGS, which stands for programmer's hierarchical interactive graphics standard. 'PHIGS', again it was then adopted by the standards organizations worldwide.

(Refer Slide Time: 09:36)

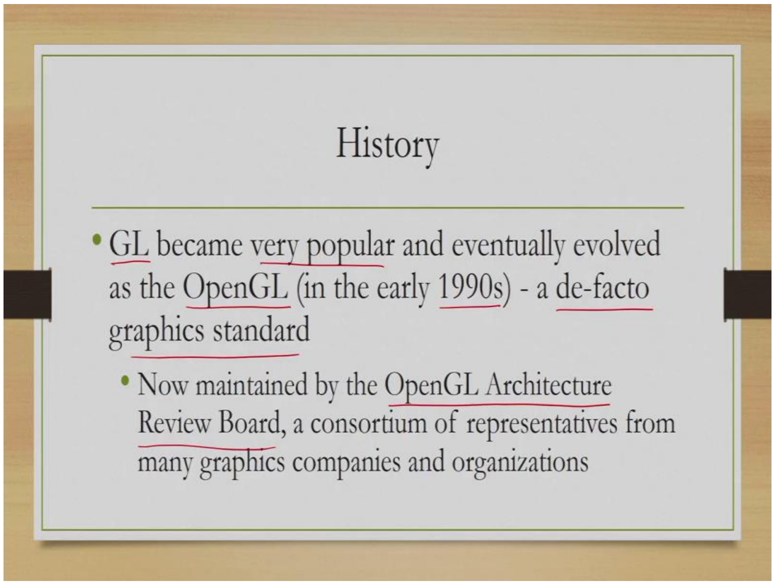


History

- Around the same time, the Silicon Graphics Inc. (SGI) started to ship their graphics workstations with a set of routines called graphics library (GL)

Now, around the same time when the other standards were being developed Silicon Graphics Inc or SGI started to ship their workstations meant for graphics with a set of routines or library functions together these are called graphics library for GL.

(Refer Slide Time: 10:10)



History

- GL became very popular and eventually evolved as the OpenGL (in the early 1990s) - a de-facto graphics standard
  - Now maintained by the OpenGL Architecture Review Board, a consortium of representatives from many graphics companies and organizations

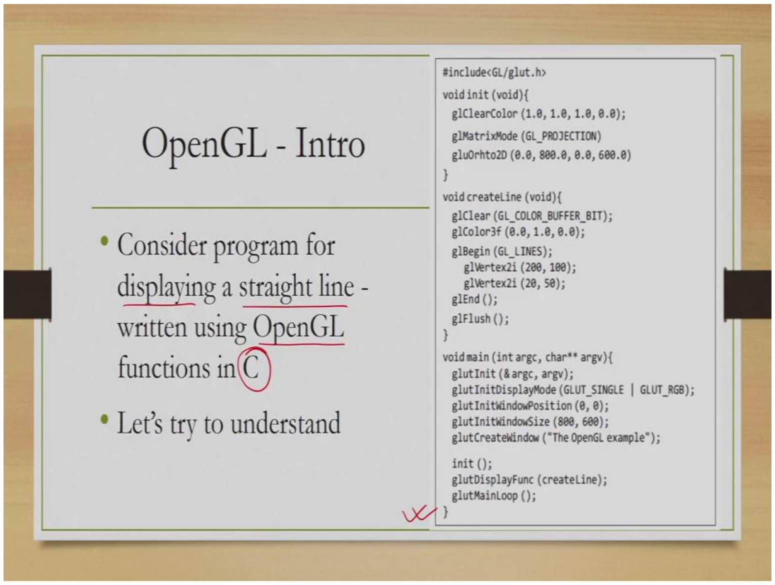
Subsequently these set of functions or GL become very popular and eventually evolved as the OpenGL in the early 1990s, which had become a de facto graphic standard. Now, this standard is now maintained by the OpenGL architecture review board which is a consortium of representatives from many graphics companies and organizations.

(Refer Slide Time: 10:45)

Programming with OpenGL

Now, let us try to understand what is there in OpenGL, what functions it provide and how to use those functions.

(Refer Slide Time: 10:58)



OpenGL - Intro

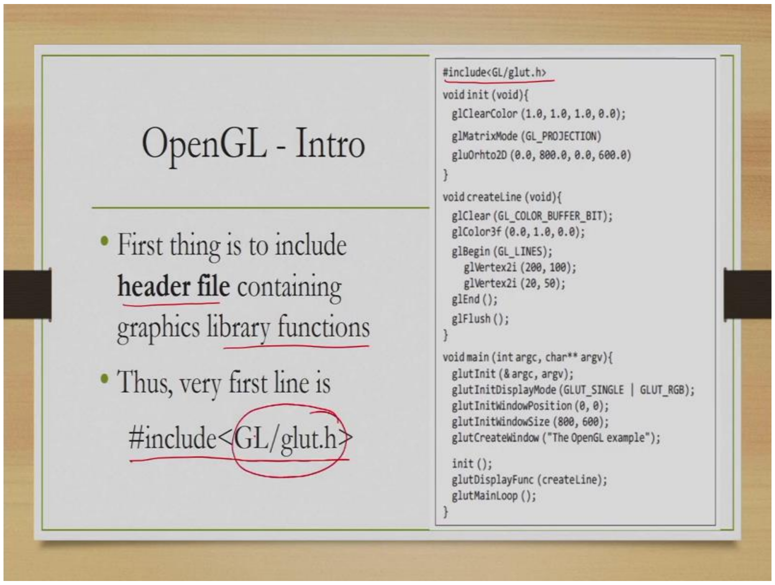
- Consider program for displaying a straight line - written using OpenGL functions in C
- Let's try to understand

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Let us try to understand OpenGL with respect to one example program. So, this program is shown here, this program is meant to display a straight line on the screen. Now, this has been written by utilizing OpenGL library functions called from C, the C language. Now, let us try to understand the syntax of the program.

(Refer Slide Time: 11:30)



OpenGL - Intro

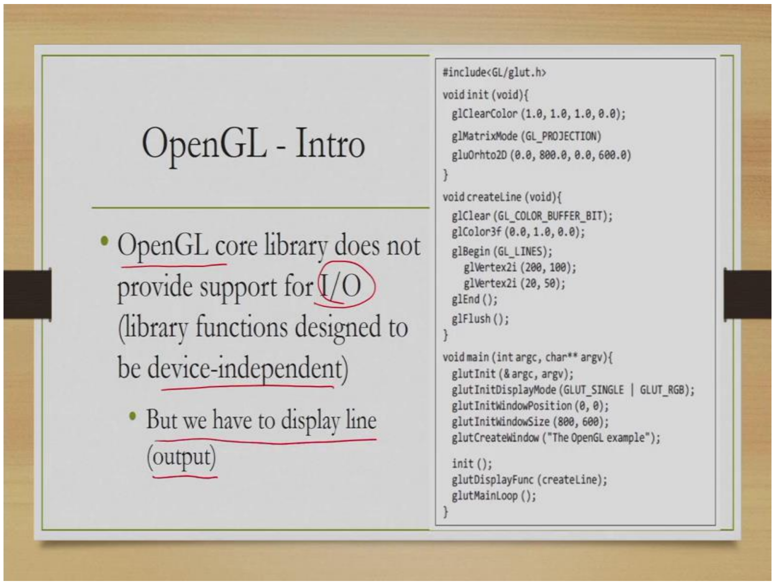
- First thing is to include **header file** containing graphics library functions
- Thus, very first line is #include<GL/glut.h>

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

So, in order to make use of the library functions, the first thing we should do is to include a header file. Now, this header file contains the library functions, so here we have included it with this statement hash include GL slash glut dot h. Now, what this library name means?

(Refer Slide Time: 12:04)



OpenGL - Intro

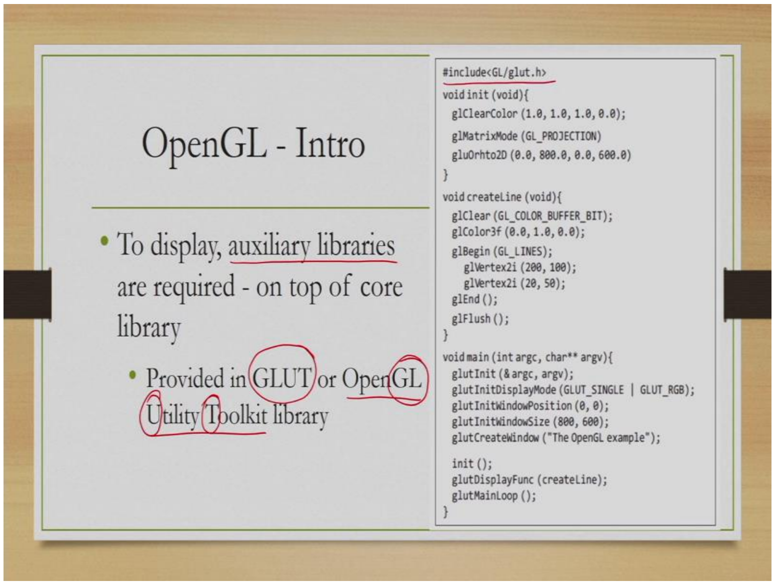
- OpenGL core library does not provide support for I/O (library functions designed to be device-independent)
  - But we have to display line (output)

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

The core library of OpenGL actually does not support input and output operations because those functions were designed to be device independent, whereas support for I/O is or must be device dependent. So, we need to do something about it because we have to display the line on the output which is essentially a device dependent operations.

(Refer Slide Time: 12:40)



OpenGL - Intro

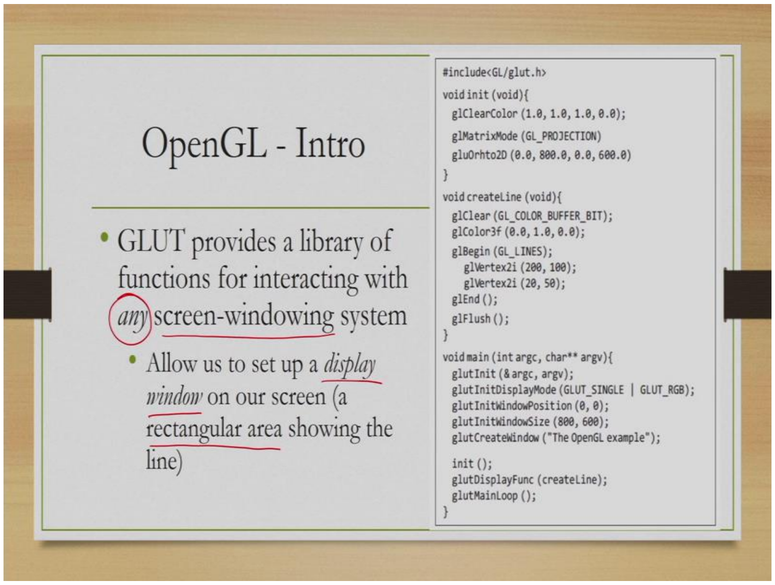
- To display, auxiliary libraries are required - on top of core library
  - Provided in GLUT or OpenGL Utility Toolkit library

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

So, to display we require auxiliary libraries on top of the code library, this is provided by the library GLUT or OpenGL utility toolkit library, ‘GLUT’, GLUT library, that is mentioned in this include statement.

(Refer Slide Time: 13:14)



OpenGL - Intro

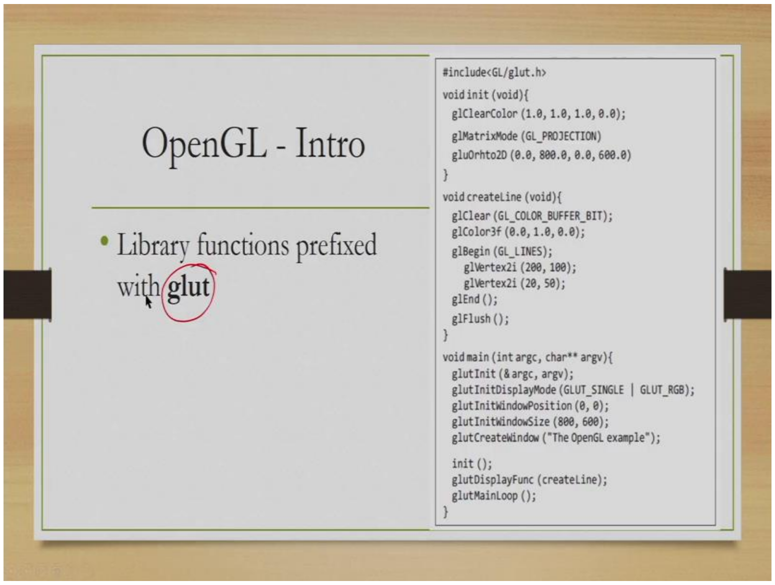
- GLUT provides a library of functions for interacting with *any* screen-windowing system
  - Allow us to set up a *display window* on our screen (a rectangular area showing the line)

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, GLUT provides a library of functions for interacting with any screen windowing system essentially any display device and it allows us to setup a display window on our screen, in this window we are going to show the image or whatever we want to display and this display window is essentially a rectangular area which contains the image, that we can do with the help of functions provided in the GLUT library.

(Refer Slide Time: 14:01)



OpenGL - Intro

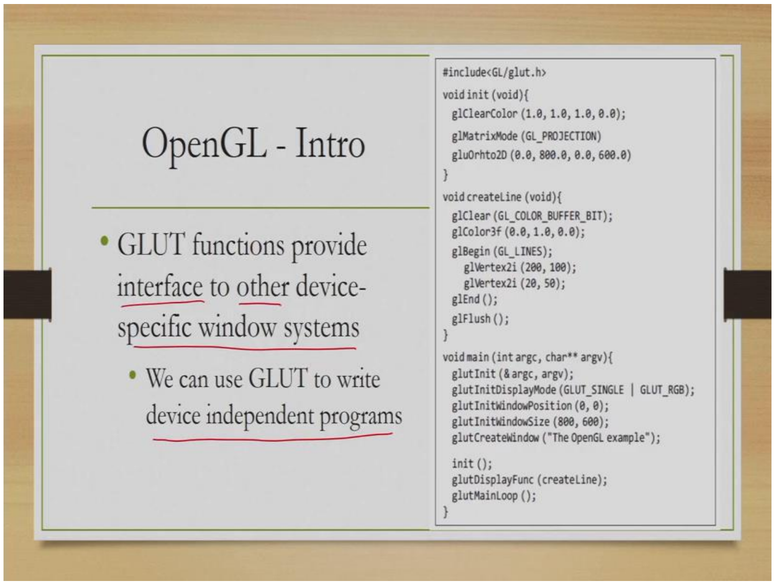
- Library functions prefixed with **glut**

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, whichever library functions we use that are part of GLUT they come with the prefix ‘glut’.

(Refer Slide Time: 14:16)



OpenGL - Intro

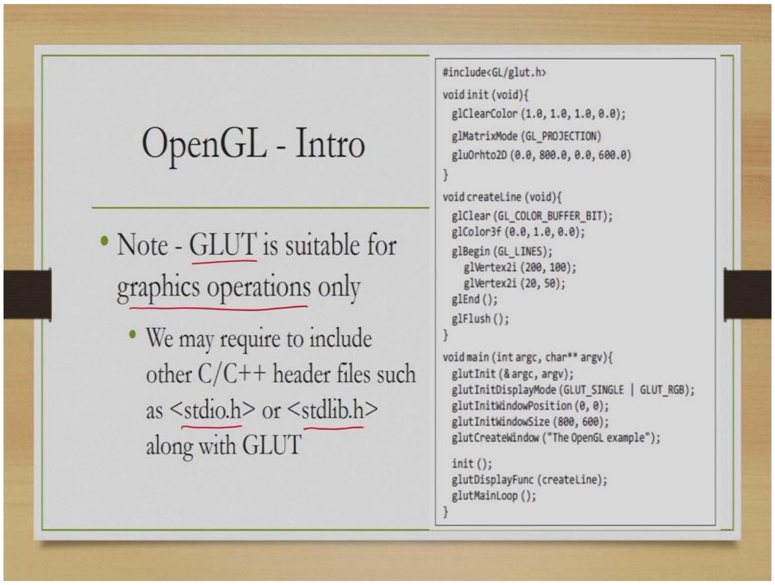
- GLUT functions provide interface to other device-specific window systems
  - We can use GLUT to write device independent programs

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

So, essentially these functions provide interface to other device specific window systems that we have already mentioned. So, we can write device independent programs using these GLUT functions and the functions themselves are used to link our program to the particular device.

(Refer Slide Time: 14:45)



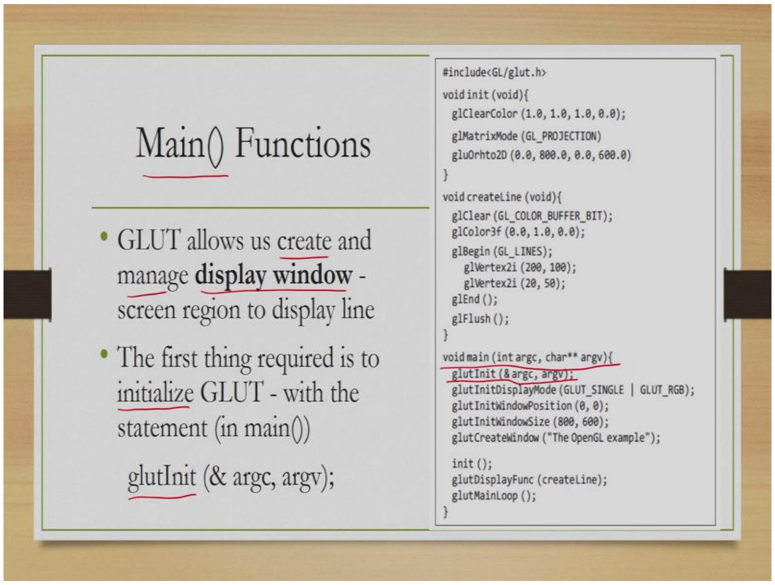
OpenGL - Intro

- Note - GLUT is suitable for graphics operations only
  - We may require to include other C/C++ header files such as <stdio.h> or <stdlib.h> along with GLUT

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Also we should note here is that the library GLUT is suitable for graphics operations only and for any other operation we may need to include other header files such as stdio.h or stdlib.h as we do in our regular programs.

(Refer Slide Time: 15:12)



Main() Functions

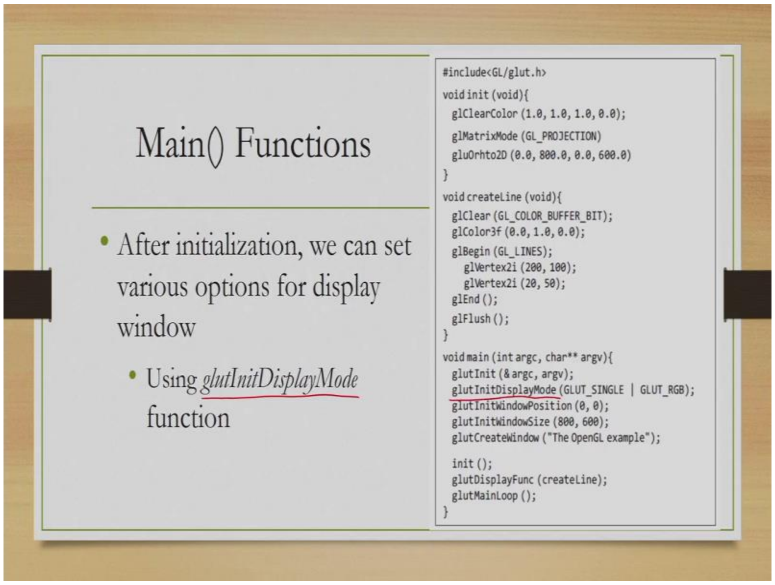
- GLUT allows us create and manage **display window** - screen region to display line
- The first thing required is to initialize GLUT - with the statement (in main())

  glutInit (& argc, argv);

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, let us start with the main function which is shown here, this function and let us try to understand the body of the function. As we said GLUT allows us to create and manage a display window or the screen region on which we want to display the line. So, the first thing that is required is to initialize GLUT with the statement glutInit as shown here, this is the initialization function that is required at the beginning.

(Refer Slide Time: 16:00)



Main() Functions

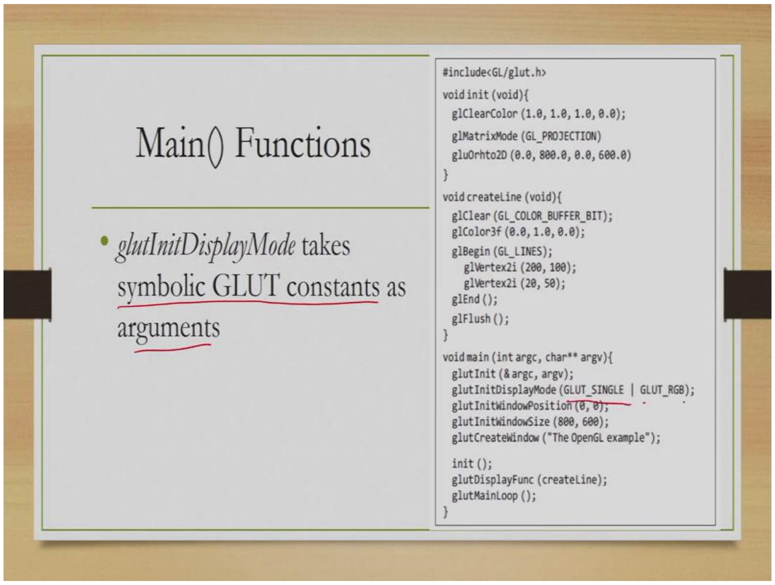
- After initialization, we can set various options for display window
  - Using *glutInitDisplayMode* function

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

After initialization, we can set various options for the display window using the function glutInitDisplayMode as shown in the second statement. So, what are these options?

(Refer Slide Time: 16:27)



Main() Functions

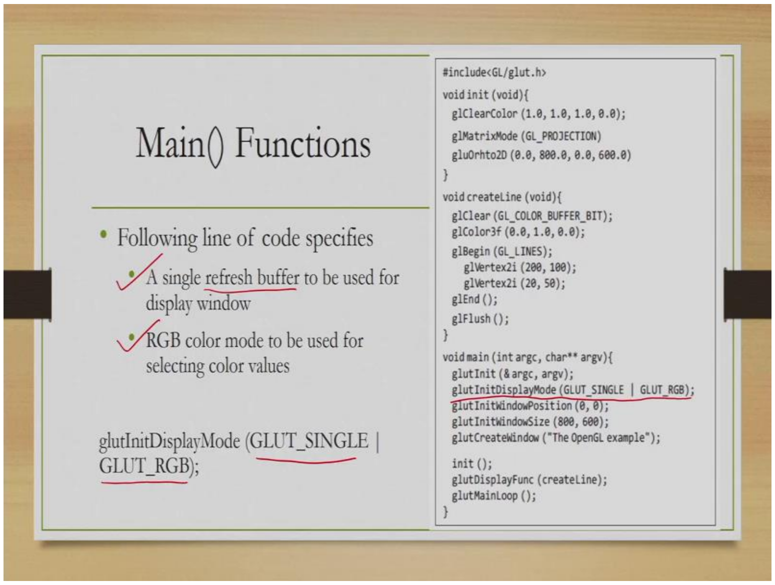
- *glutInitDisplayMode* takes symbolic GLUT constants as arguments

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, these options are provided by symbolic GLUT constants as arguments as shown here, GLUT_SINGLE, GLUT_RGB.

(Refer Slide Time: 16:44)



Main() Functions

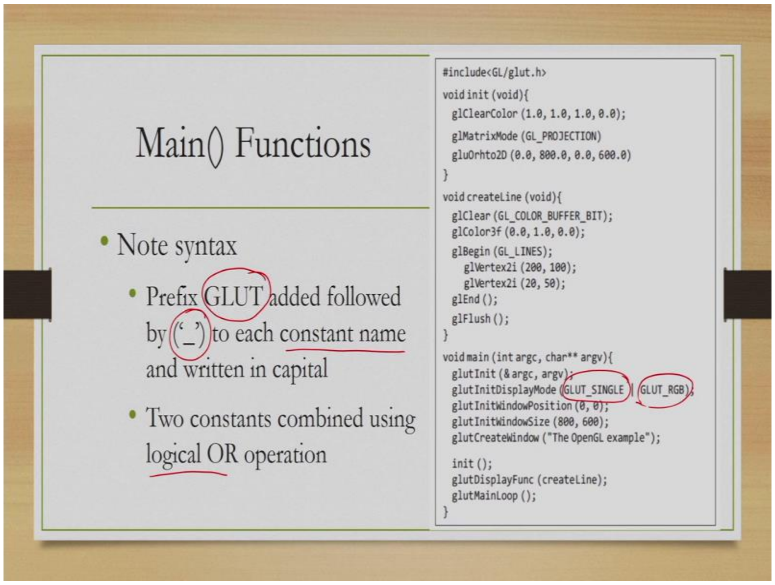
- Following line of code specifies
  - A single refresh buffer to be used for display window
  - RGB color mode to be used for selecting color values

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, here in this particular function we have used this statement having these two arguments GLUT_SINGLE and GLUT_RGB, they indicate that we are specifying a single refresh buffer to be used for the display window and RGB color mode to be used for selecting color values. GLUT_SINGLE is for the first task single refresh buffer and GLUT_RGB indicates that RGB color mode to be used.

(Refer Slide Time: 17:23)



Main() Functions

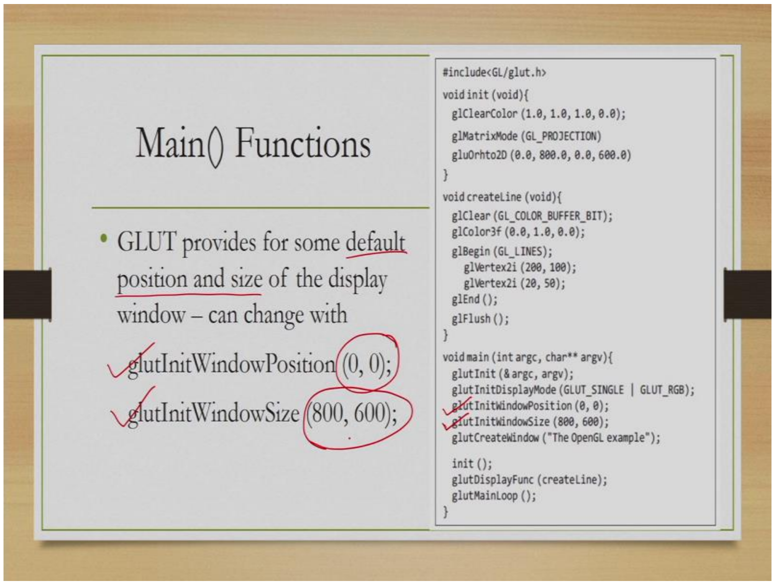
- Note syntax
  - Prefix GLUT added followed by ('_') to each constant name and written in capital
  - Two constants combined using logical OR operation

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, here we should look at the syntax, how this glutInitDisplayMode function is used. In the constant name which provides the options, we have used GLUT as a prefix all caps followed by a underline symbol and then the constant name again all caps as shown here or here this is the particular syntax used to provide arguments. Now, to combine multiple options we are using this logical or operation, to indicate that we want both that is the syntax used to provide the options.

(Refer Slide Time: 18:32)



Main() Functions

- GLUT provides for some default position and size of the display window – can change with

  glutInitWindowPosition (0, 0);

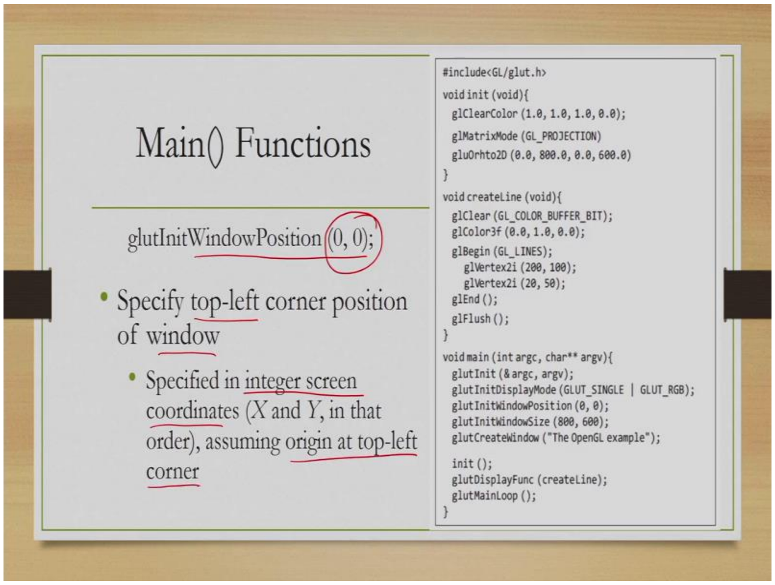
  glutInitWindowSize (800, 600);

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Then we are using the two functions glutInitwindowPosition and glutInitwindowSize. Now, these are used to provide some values that are different than the default values for the window size and position that is already there in the library function. So, if we want to change the values then we need to use these two functions glutInitwindowPosition where is specify the value and glutInitwindowSize where we specify again the size value.

(Refer Slide Time: 19:16)



Main() Functions

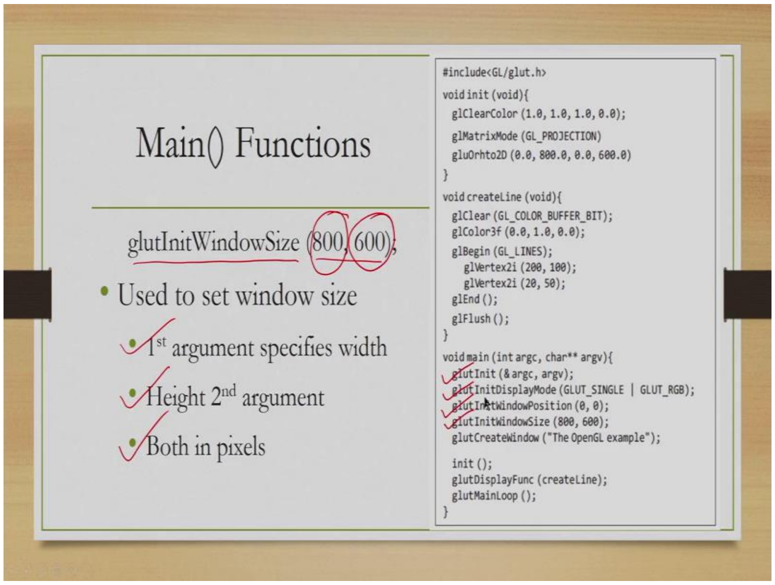
glutInitWindowPosition (0, 0);

- Specify top-left corner position of window
  - Specified in integer screen coordinates (*X* and *Y*, in that order), assuming origin at top-left corner

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, this window position, which position is specifies? It specifies top left corner position of the window. Assuming integer screen coordinate system and assuming origin at the top left corner. These are the assumptions when we specify these values.

(Refer Slide Time: 19:45)



Main() Functions

glutInitWindowSize (800, 600);

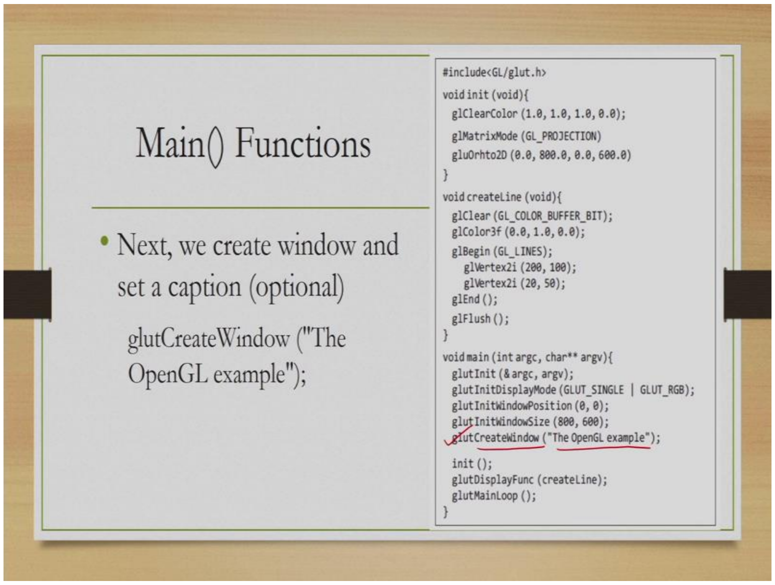
- Used to set window size
  - 1st argument specifies width
  - Height 2nd argument
  - Both in pixels

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Then in case of glutInitwindowSize where we are specifying the size, the first argument specifies width that means 800, second argument specifies height that is 600 and both these values are in pixels, so 800 pixels by 600 pixels. So, we have understood these four functions init, displaymode, windowPosition and windowSize.

(Refer Slide Time: 20:26)



Main() Functions

- Next, we create window and set a caption (optional)

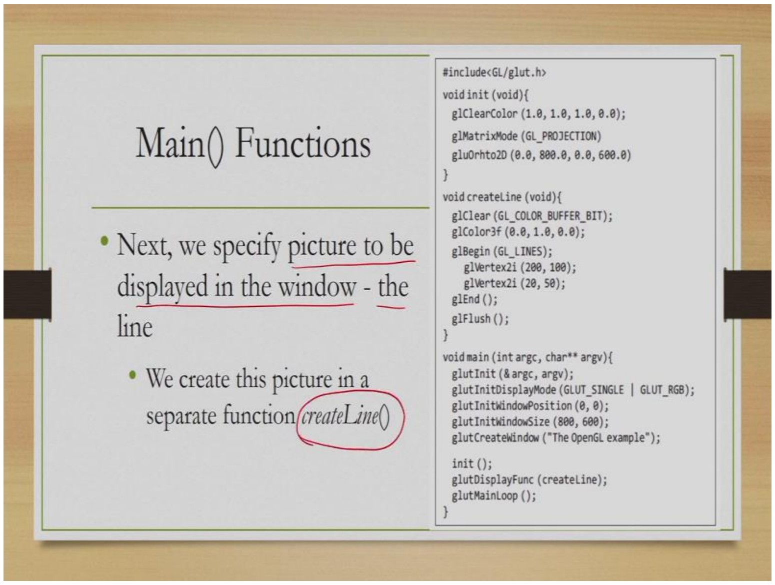
  glutCreateWindow ("The OpenGL example");

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Next we create the window and set a caption which is optional using the function Createwindow and the caption is provided within parentheses, but this caption is optional.

(Refer Slide Time: 20:52)



Main() Functions

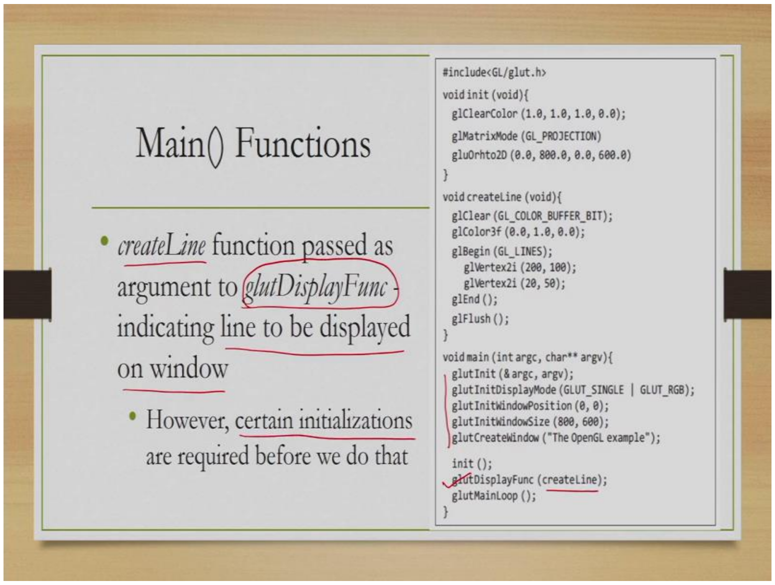
- Next, we specify picture to be displayed in the window - the line
  - We create this picture in a separate function createLine()

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

The next thing that we do is specify that the picture is to be displayed in the window that is the line. Now, we have to create the line and then we can display it in the window, this creation is done by a separate function which is user defined which we are calling createLine function.

(Refer Slide Time: 21:23)



Main() Functions

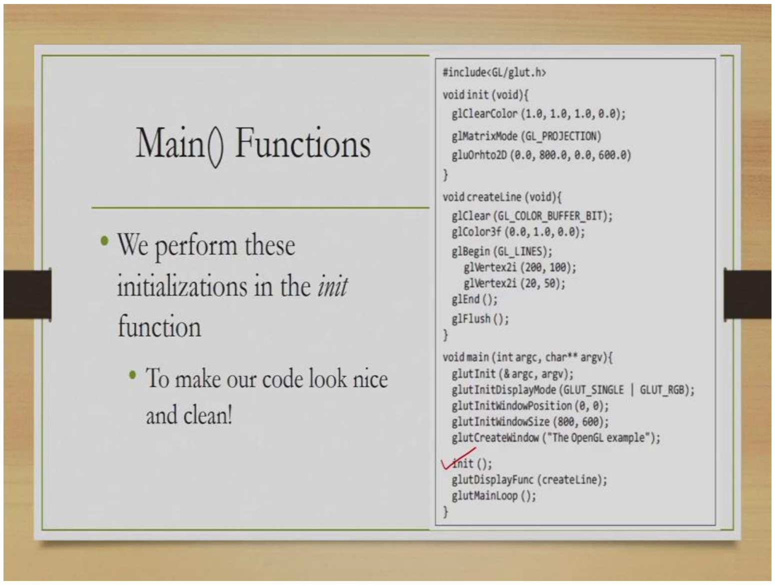
- createLine function passed as argument to glutDisplayFunc indicating line to be displayed on window
  - However, certain initializations are required before we do that

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, this createLine function is passed as an argument to another glut library function that is glutDisplayFunction which is shown here. This indicates that the line is to be displayed on the window. So, with this function we indicate that we are creating a line which is our image here that is using the create line function and this line is to be displayed on the window created through these statements. But before we do that certain initializations are required.

(Refer Slide Time: 22:05)



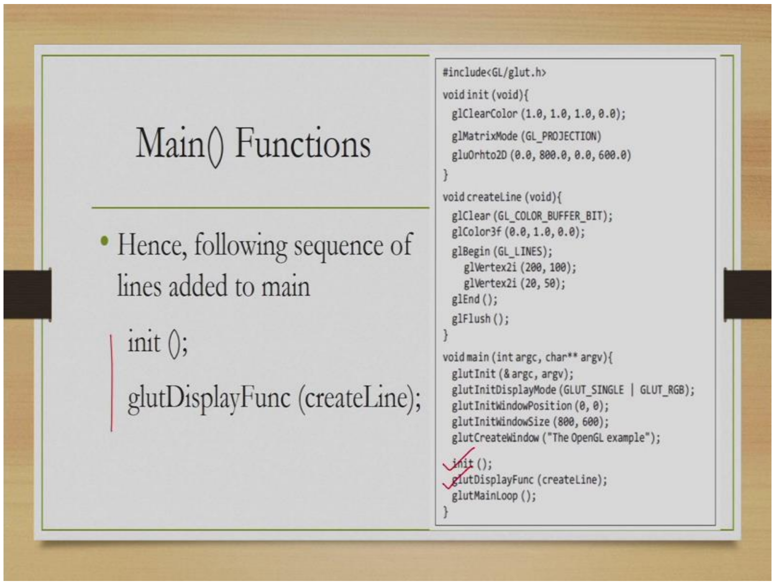
Main() Functions

- We perform these initializations in the *init* function
  - To make our code look nice and clean!

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

And these initializations are performed in the init function shown here. Again this init function is used to make our code look very clean, otherwise we could have used it in a different way and will come back to this function later.

(Refer Slide Time: 22:37)



Main() Functions

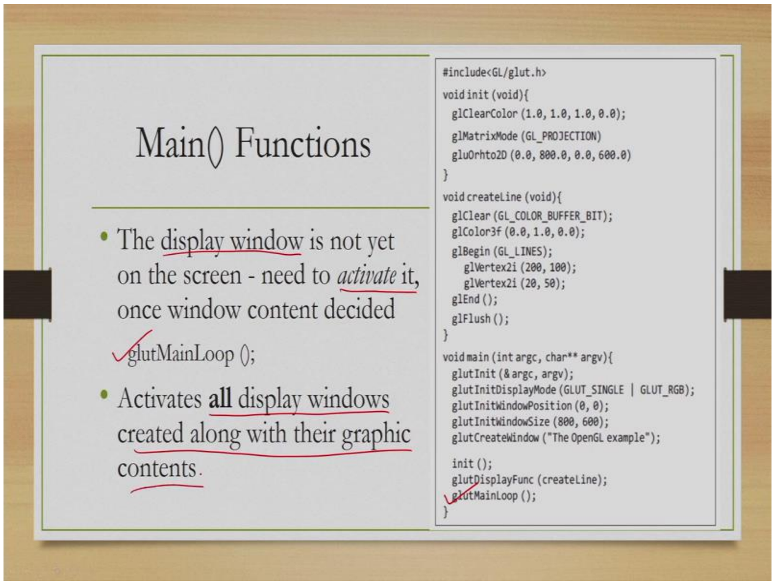
- Hence, following sequence of lines added to main

```
init ();
glutDisplayFunc (createLine);
```

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

So, in order to keep the code clean and to indicate that we want to display a line on the window we add these two lines init and glutDisplayFunction as shown here.

(Refer Slide Time: 22:53)



Main() Functions

- The display window is not yet on the screen - need to *activate* it, once window content decided

glutMainLoop ();

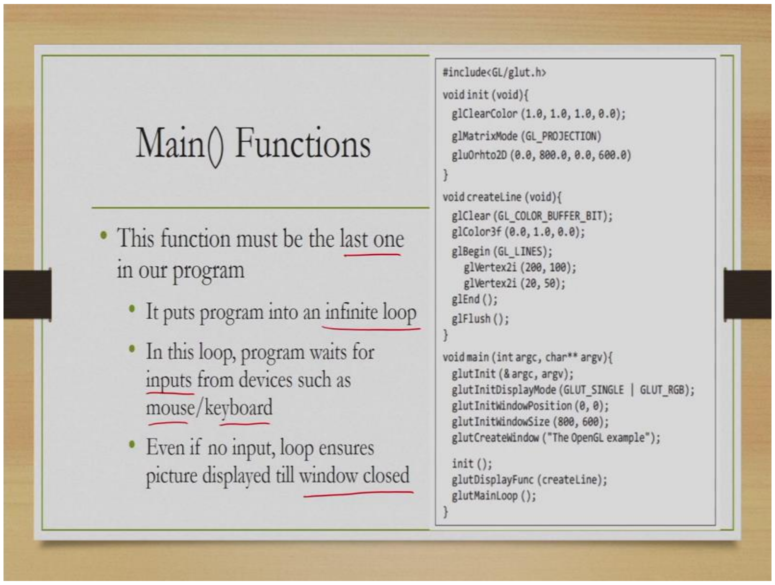
- Activates **all** display windows created along with their graphic contents.

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, those are all done but the window is still not on the screen, we need to activate it once the window content is decided, that we do with this function glutMainLoop. Here it activates all display windows created along with their graphic contents. So, this function glutMainLoop actually puts the window with its content on the screen.

(Refer Slide Time: 23:29)



Main() Functions

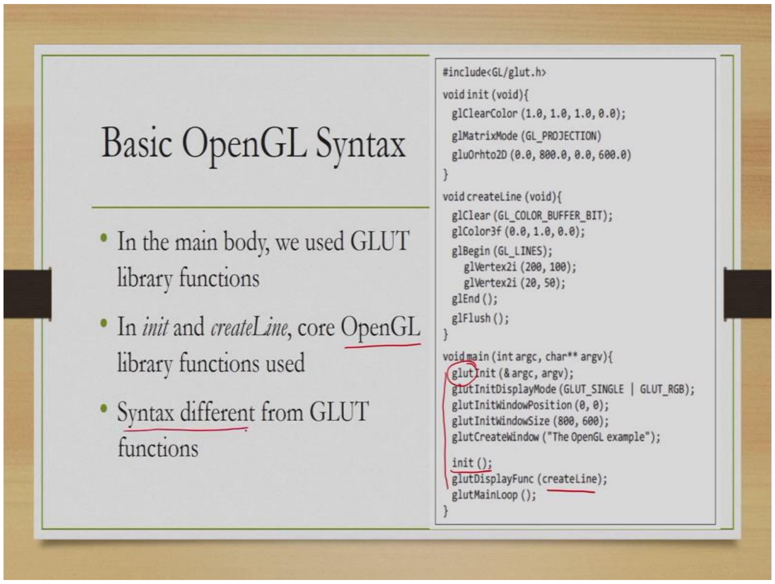
- This function must be the last one in our program
  - It puts program into an infinite loop
  - In this loop, program waits for inputs from devices such as mouse/keyboard
  - Even if no input, loop ensures picture displayed till window closed

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

This function must be the last one in our program, it puts the program into an infinite loop because the display we want constantly. In this loop the program waits for inputs from devices an input device such as mouse, keyboard, even if there is no input the loop ensures that the picture is displayed till the window is closed. So, since we want the picture to remain on the screen unless there is some input or the window is closed we use the loop and this loop must be at the last statement of the code in main after we create the image and put it on the window.

(Refer Slide Time: 24:23)



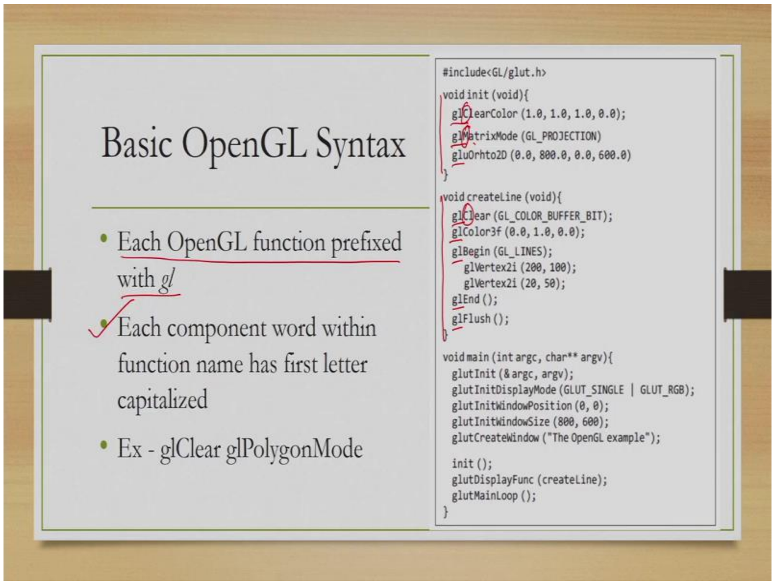
Basic OpenGL Syntax

- In the main body, we used GLUT library functions
- In *init* and *createLine*, core OpenGL library functions used
- Syntax different from GLUT functions

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, as we have noted so we explained all these functions that are there in main and all this started with glut indicating that there glut library function except the two functions init and create line. Now, in these two functions we used OpenGL library function rather than glut library functions accordingly their syntax are different.

(Refer Slide Time: 24:57)



Basic OpenGL Syntax

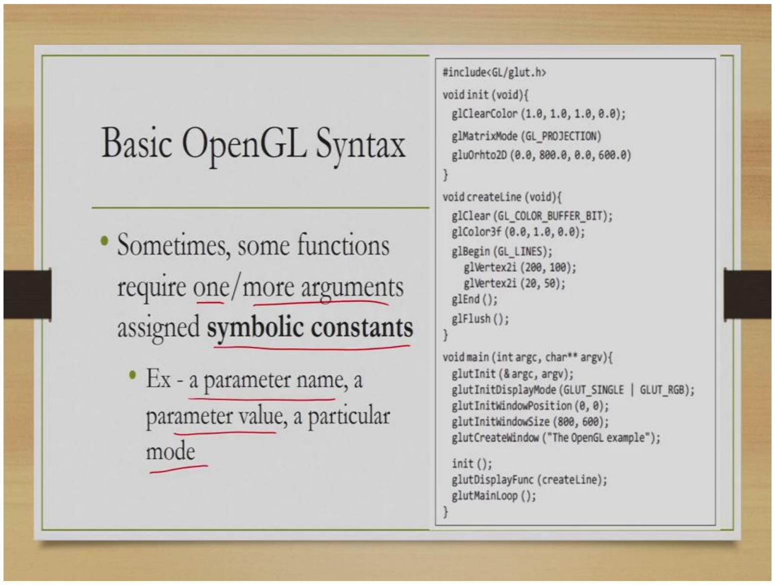
- Each OpenGL function prefixed with *gl*
- Each component word within function name has first letter capitalized
- Ex - glClear glPolygonMode

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Each OpenGL function prefixed with GL as we can see in this function init as well as in this create line function. So, here each function is starting with this prefix gl, it indicates that this function is a OpenGL function. Each component word within the function name has first letter capitalized like here C is capitalized in all the cases as you can see Matrix M is capitalized and so on. So, that is the syntax of OpenGL library function starts with gl and component word within this function name has first letter capitalized.

(Refer Slide Time: 25:57)



Basic OpenGL Syntax

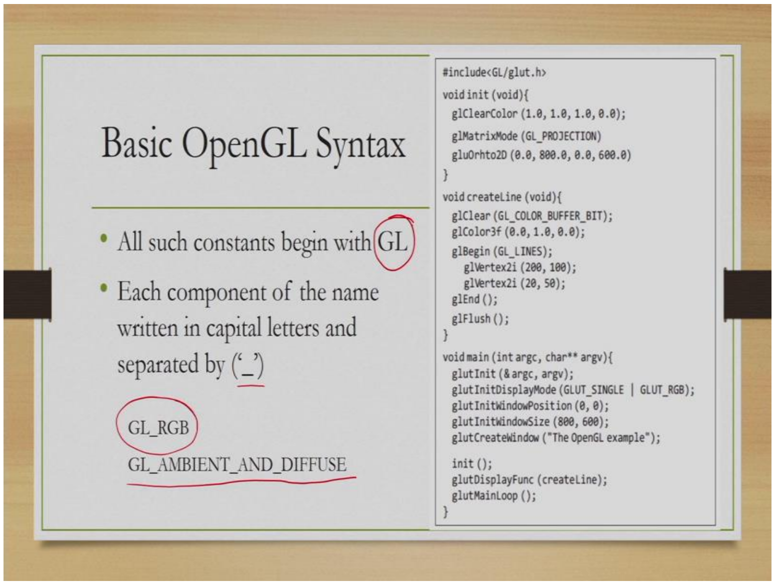
- Sometimes, some functions require one/more arguments assigned **symbolic constants**
  - Ex - a parameter name, a parameter value, a particular mode

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Sometimes some functions may require one or more arguments which are assigned symbolic constants. For example, a parameter name, parameter value or a particular mode, these are all part of the OpenGL library function syntax.

(Refer Slide Time: 26:21)



Basic OpenGL Syntax

- All such constants begin with GL
- Each component of the name written in capital letters and separated by ('_')

GL_RGB

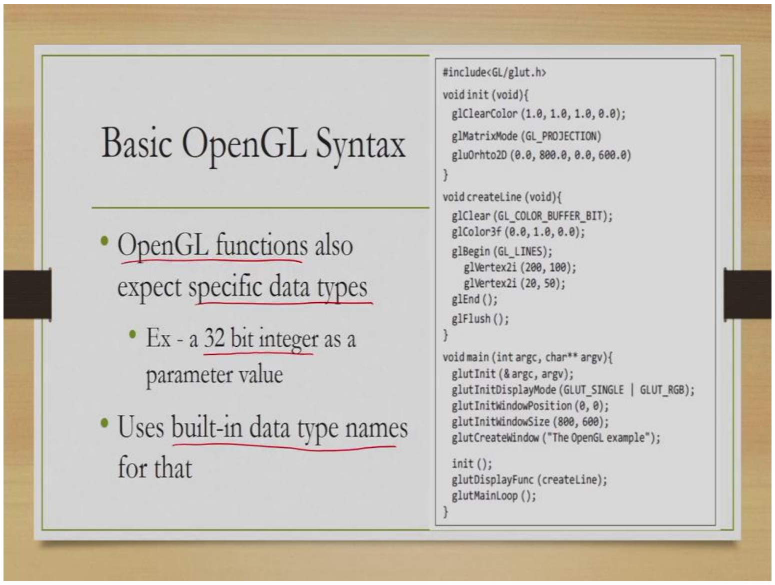
GL_AMBIENT_AND_DIFFUSE

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, all these constants begin with capital GL all in capital. Each component of the name is written in capital letters and separated by underline symbol, as we have seen in the case of glut constants as well like GL underscore RGB, GL underscore AMBIENT underscore AND underscore DIFFUSE, where everything is in capital separated by underline.

(Refer Slide Time: 26:57)



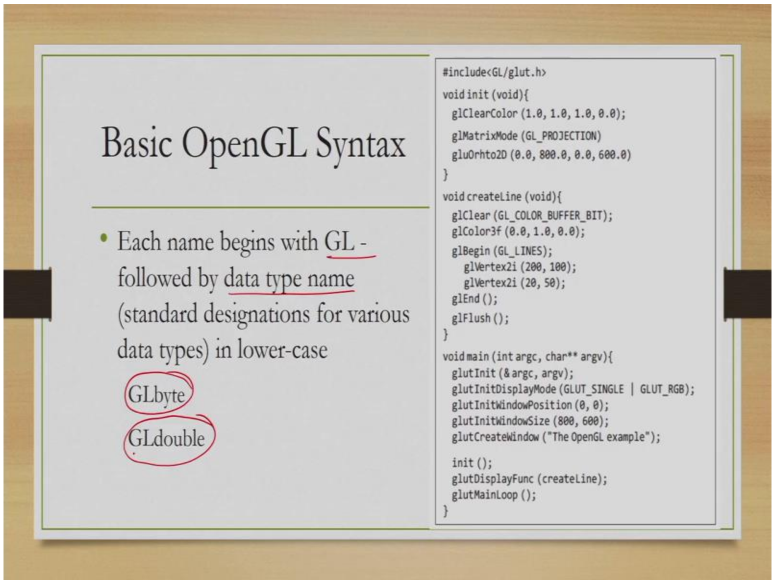
Basic OpenGL Syntax

- OpenGL functions also expect specific data types
  - Ex - a 32 bit integer as a parameter value
- Uses built-in data type names for that

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Also the OpenGL functions expect some specific data types. For example, 32 bit integer as a parameter value and these functions use built-in data type names for that.

(Refer Slide Time: 27:19)



Basic OpenGL Syntax

- Each name begins with GL - followed by data type name (standard designations for various data types) in lower-case

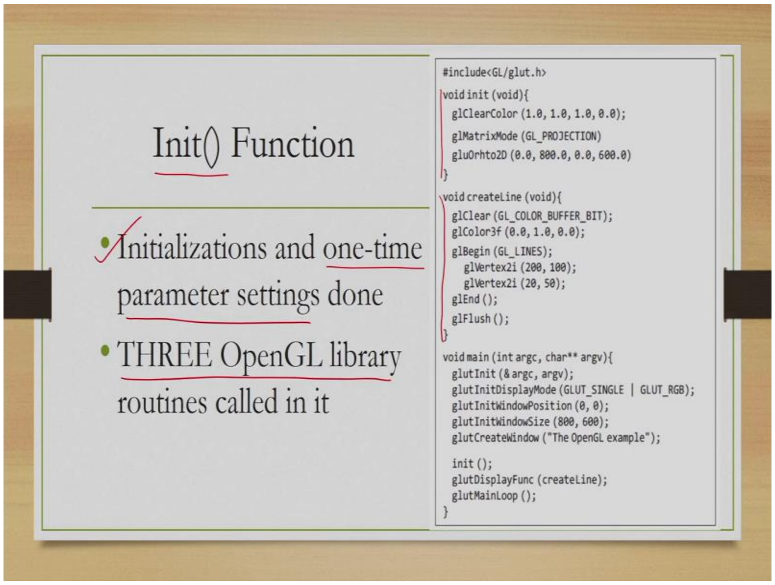
  GLbyte

  GLdouble

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Each of these names begins with GL and followed by data type name. For example, GLbyte, GLdouble, but this data type name is in lowercase.

(Refer Slide Time: 27:42)



Init() Function

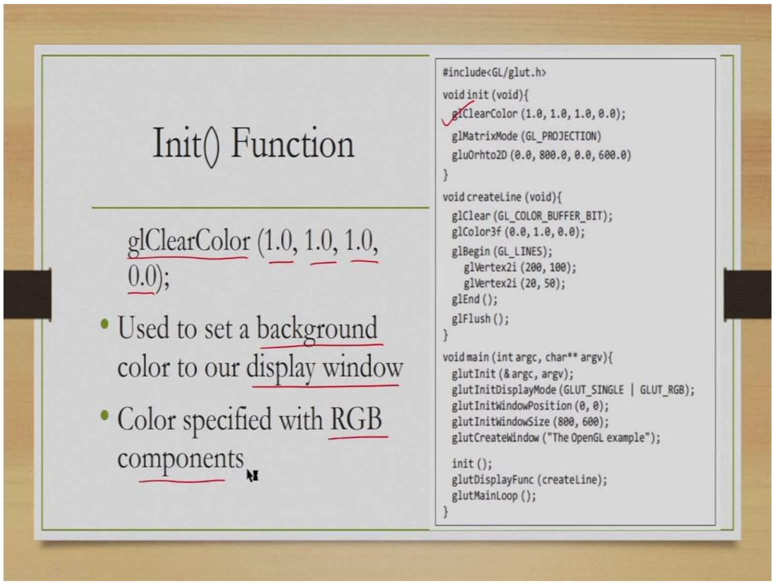
- Initializations and one-time parameter settings done
- THREE OpenGL library routines called in it

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

So, those are the syntax that are used for using OpenGL library functions. Now, let us try to understand these two functions that we have defined using the OpenGL library functions, one is init, one is create line. So, let us start with init. This is essentially mean to initialize and perform one time parameter settings. In our function we have used three OpenGL library routines or library functions. What they do?

(Refer Slide Time: 28:25)



Init() Function

glClearColor (1.0, 1.0, 1.0, 0.0);

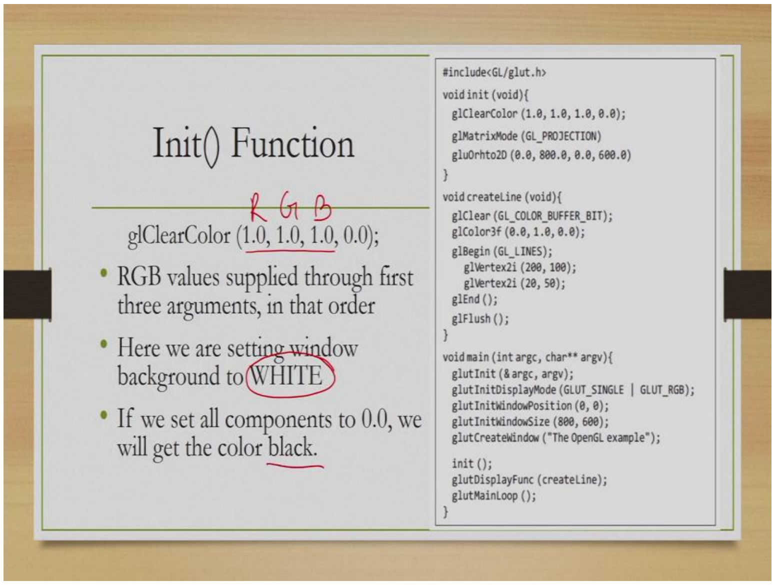
- Used to set a background color to our display window
- Color specified with RGB components

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, one is glClearColor, the first one with some argument, four arguments are used. This is used to set a background color to our display window and this color is specified with RGB components.

(Refer Slide Time: 28:48)



Init() Function

R G B

glClearColor (1.0, 1.0, 1.0, 0.0);

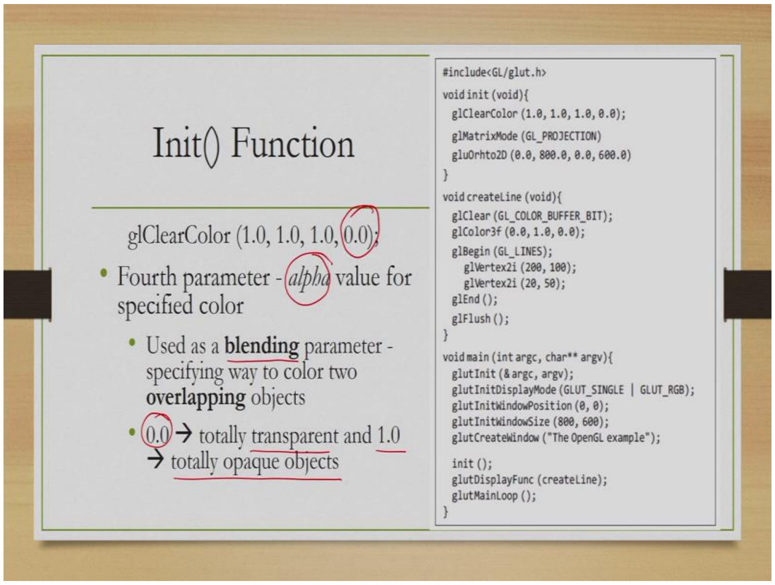
- RGB values supplied through first three arguments, in that order
- Here we are setting window background to WHITE
- If we set all components to 0.0, we will get the color black.

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, these RGB components are specified in the first three arguments in that order that means this is R, this is G, this is B, with this particular set of values as we all know you will get white as the background color, we can set any background color. For example, if we set all 0, we will get black.

(Refer Slide Time: 29:21)



Init() Function

glClearColor (1.0, 1.0, 1.0, 0.0);

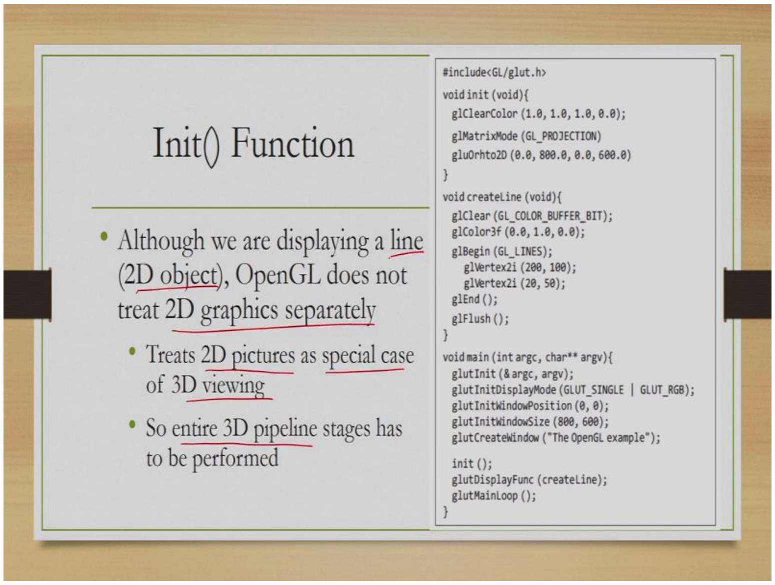
- Fourth parameter - *alpha* value for specified color
  - Used as a **blending** parameter - specifying way to color two **overlapping** objects
  - 0.0 → totally transparent and 1.0 → totally opaque objects

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, there is also a fourth parameter which we have set as 0.0. Now, this is called alpha value for the specified color and it is used as a blending parameter. In other words it specifies transparency of the color. If we are using value 0.0 that means the color is totally transparent and 1.0 means totally opaque objects. So, it indicates transparency.

(Refer Slide Time: 30:08)



Init() Function

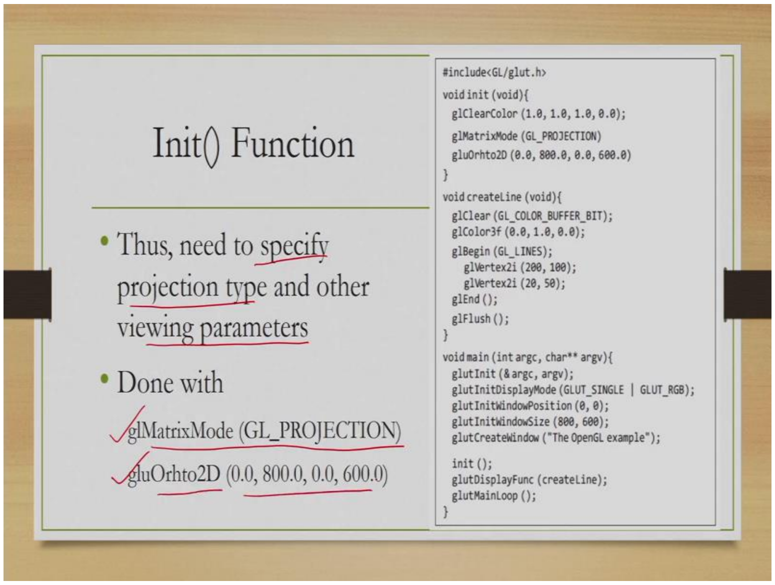
- Although we are displaying a line (2D object), OpenGL does not treat 2D graphics separately
  - Treats 2D pictures as special case of 3D viewing
  - So entire 3D pipeline stages has to be performed

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, here we are displaying a line which is a 2D object. However, OpenGL does not treat 2D objects separately. Now, it treats the 2D pictures as special case of 3D viewing. So, essentially the entire 3D pipeline stages are performed.

(Refer Slide Time: 30:34)



Init() Function

- Thus, need to specify projection type and other viewing parameters
- Done with

✓glMatrixMode (GL_PROJECTION)

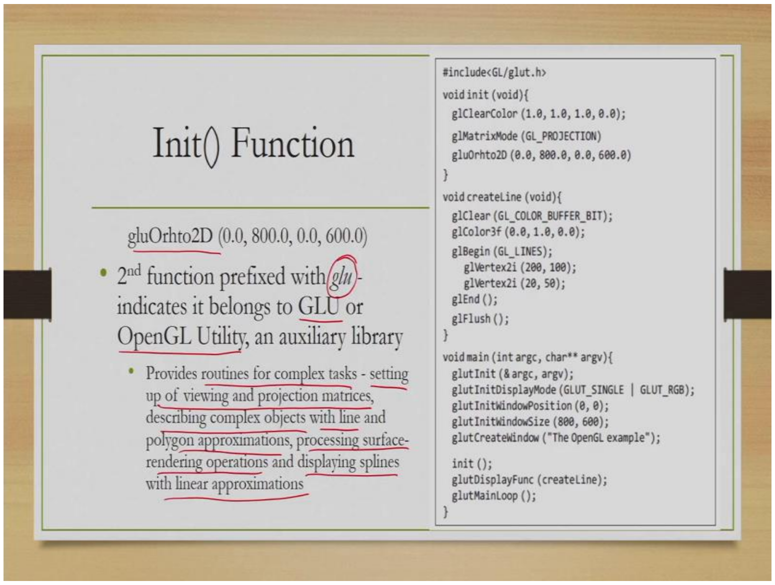
✓gluOrhto2D (0.0, 800.0, 0.0, 600.0)

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

So, we need to specify the projection type and other viewing parameters that is done with these two functions glMatrixMode, which is GL_PROJECTION and gluOrtho2D with some arguments.

(Refer Slide Time: 30:58)



Init() Function

gluOrhto2D (0.0, 800.0, 0.0, 600.0)

- 2nd function prefixed with glu - indicates it belongs to GLU or OpenGL Utility, an auxiliary library
  - Provides routines for complex tasks - setting up of viewing and projection matrices, describing complex objects with line and polygon approximations, processing surface-rendering operations and displaying splines with linear approximations

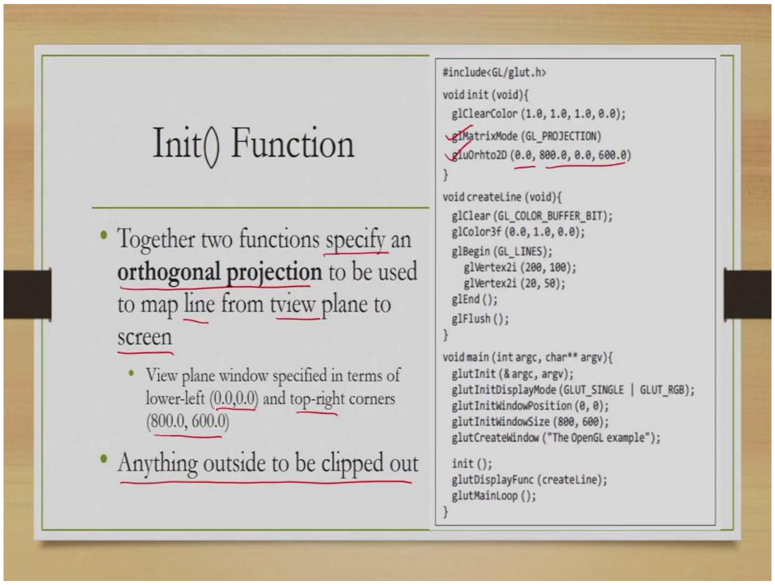
```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, this function gluOrtho2D here the function is prefixed to GLU rather than GL. So, it indicates that this function belongs to GLU or OpenGL utility another auxiliary library. Earlier we have seen GLUT OpenGL utility toolkit, now we are seeing OpenGL utility another auxiliary library.

And this library provides routines for complex tasks such as setting up of viewing and projection matrices, describing complex objects with line and polygon approximations, processing of surface rendering operations and displaying splines with linear approximations, these are some examples of the complex tasks that are part of the pipeline which are implemented in this OpenGL utility auxiliary library.

(Refer Slide Time: 32:00)



Init() Function

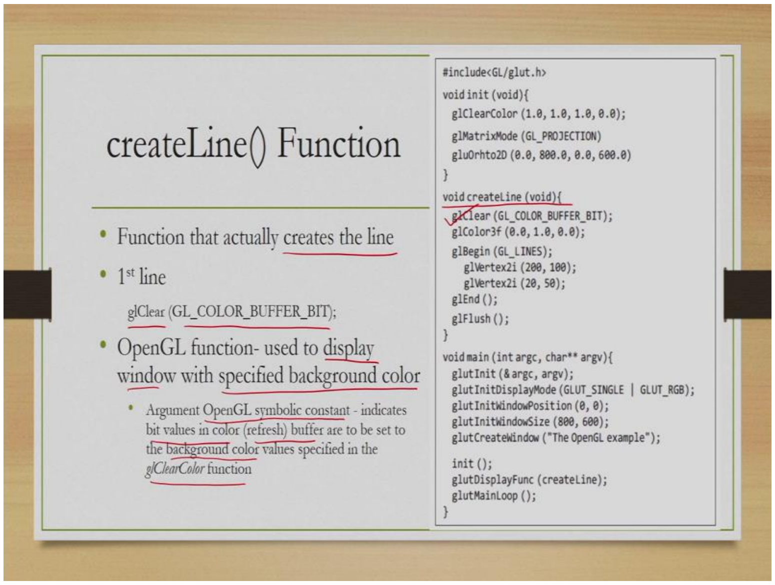
- Together two functions specify an **orthogonal projection** to be used to map line from tview plane to screen
  - View plane window specified in terms of lower-left (0.0,0.0) and top-right corners (800.0, 600.0)
- Anything outside to be clipped out

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, together these two functions glMatrixMode and gluOrtho2D specify an orthogonal projection to be used to map the line from view plane to the screen. Now, view plane window specified in terms of lower left and top right corners of the window. So, these arguments specify the lower left and top right corners of the window and during this projection anything outside of this window is clipped out as we have discussed during our pipeline discussion.

(Refer Slide Time: 32:58)



createLine() Function

- Function that actually creates the line
- 1st line

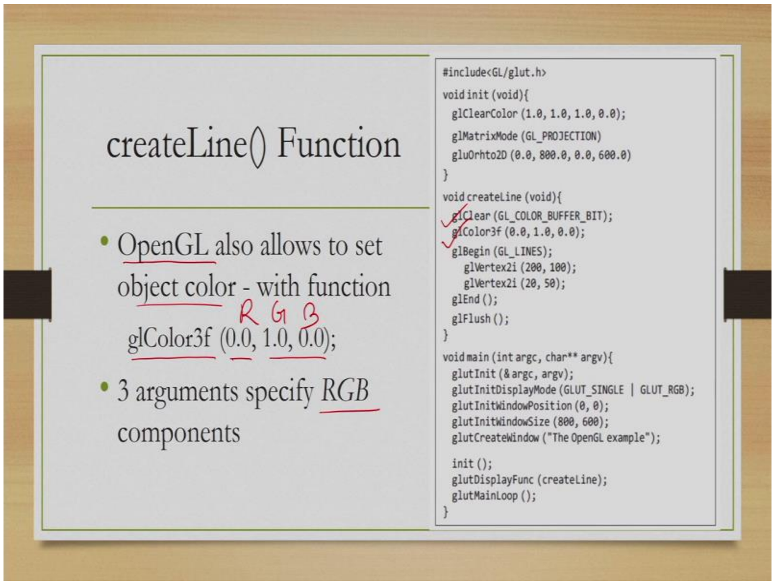
  glClear (GL_COLOR_BUFFER_BIT);
- OpenGL function- used to display window with specified background color
  - Argument OpenGL symbolic constant - indicates bit values in color (refresh) buffer are to be set to the background color values specified in the *glClearColor* function

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, let us move to our second function create line. Now, this function is actually creates the line which we want to display. The first line is glClear with some arguments. Now, this function is used to display the window with specified background color. Now, the arguments as you can see an OpenGL symbolic constant indicates bit values in color or the refresh buffer that are to be set to the background color values specified in the function glClearColor function. So, essentially this function indicates what should be the background color of the display window.

(Refer Slide Time: 34:10)



createLine() Function

- OpenGL also allows to set object color - with function

  R G B

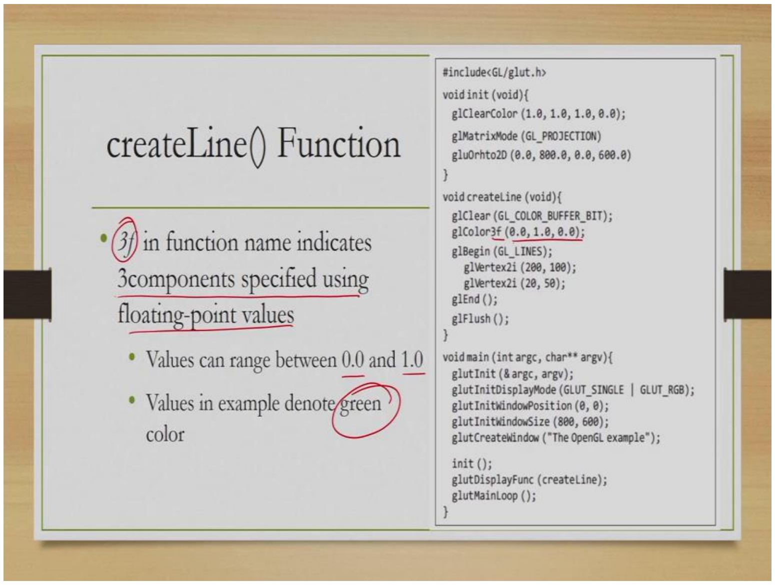
  glColor3f (0.0, 1.0, 0.0);
- 3 arguments specify *RGB* components

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, OpenGL function also allows us to set the object color with the function glColor3f. So, there are three arguments again they specify the RGB components 'RGB', so these two functions are used to set color values to the background as well as to the object.

(Refer Slide Time: 34:37)



createLine() Function

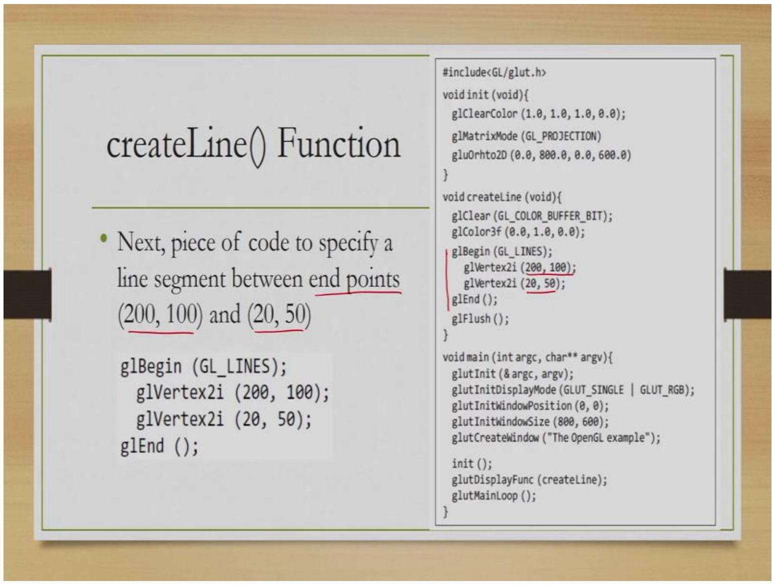
- 3f in function name indicates 3components specified using floating-point values
  - Values can range between 0.0 and 1.0
  - Values in example denote green color

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, in the second function this 3f indicates that the three components are specified using floating point values. In other words the values can range between 0.0 to 1.0. So, in this particular case these three values denote green color.

(Refer Slide Time: 35:09)



createLine() Function

- Next, piece of code to specify a line segment between end points (200, 100) and (20, 50)

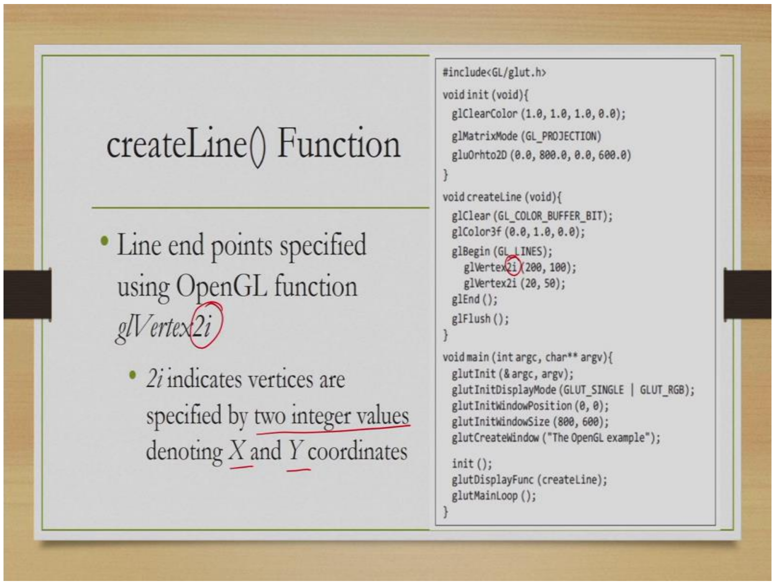
```
glBegin (GL_LINES);
  glVertex2i (200, 100);
  glVertex2i (20, 50);
glEnd ();
```

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Next we have a piece of code between the two functions glBegin and glEnd, so this indicates the line segment to be drawn between the endpoints provided in the arguments. So, this function essentially creates the line between these two endpoints specified with these two arguments and the functions glVertex2i called twice.

(Refer Slide Time: 35:50)



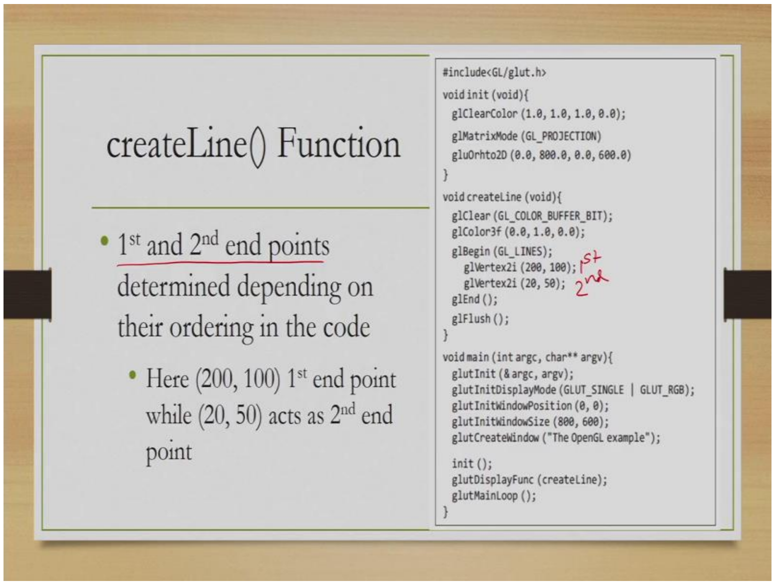
createLine() Function

- Line end points specified using OpenGL function *glVertex2i*
  - *2i* indicates vertices are specified by two integer values denoting *X* and *Y* coordinates

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, here this 2i in the function as you can guess indicates that the vertices are specified by two integer values denoting the X and Y coordinates, this is quite straightforward.

(Refer Slide Time: 36:10)



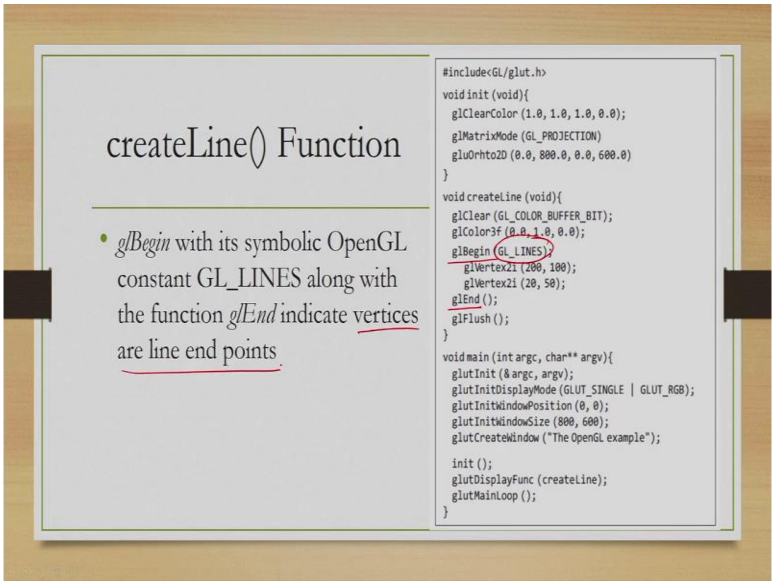
createLine() Function

- 1st and 2nd end points determined depending on their ordering in the code
  - Here (200, 100) 1st end point while (20, 50) acts as 2nd end point

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100); 1st
    glVertex2i (20, 50);  2nd
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");
  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, the first and second endpoints are determined depending on their ordering in the code. So, this will always be treated as the first point because it is appearing before the other one and this will be the second point. So, in the way the code is written the first and second points are determined.

(Refer Slide Time: 36:35)



createLine() Function

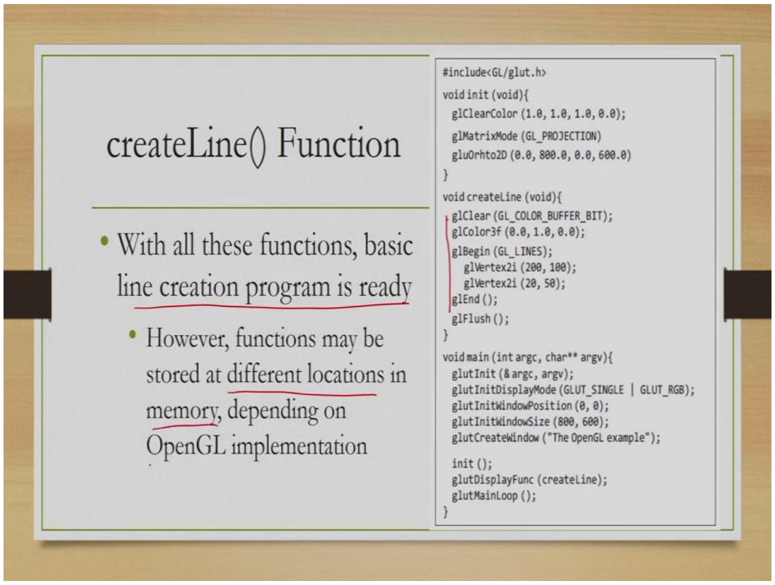
- *glBegin* with its symbolic OpenGL constant GL_LINES along with the function *glEnd* indicate vertices are line end points

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

And this function glBegin with this constant GL_LINES as well as the function glEnd indicate that the vertices are line end points.

(Refer Slide Time: 36:56)



createLine() Function

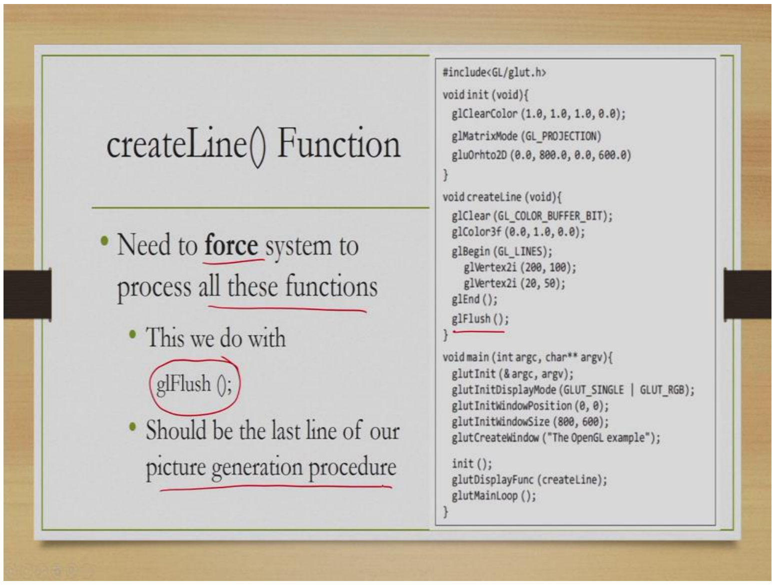
- With all these functions, basic line creation program is ready
  - However, functions may be stored at different locations in memory, depending on OpenGL implementation

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

Now, with all these functions our basic line creation program is ready. One point to be noted here is that these functions may be stored at different locations in the memory depending on the way OpenGL is implemented.

(Refer Slide Time: 37:19)



createLine() Function

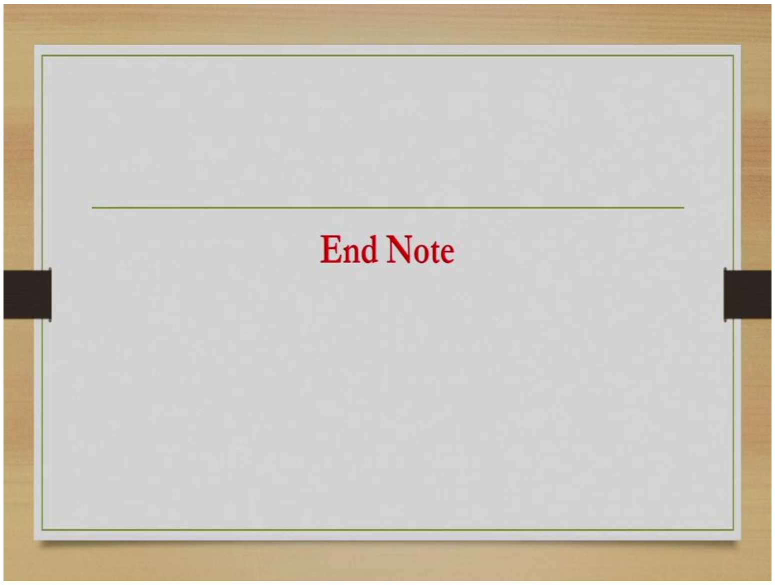
- Need to **force** system to process all these functions
  - This we do with glFlush ();
  - Should be the last line of our picture generation procedure

```
#include<GL/glut.h>
void init (void){
  glClearColor (1.0, 1.0, 1.0, 0.0);
  glMatrixMode (GL_PROJECTION)
  gluOrhto2D (0.0, 800.0, 0.0, 600.0)
}
void createLine (void){
  glClear (GL_COLOR_BUFFER_BIT);
  glColor3f (0.0, 1.0, 0.0);
  glBegin (GL_LINES);
    glVertex2i (200, 100);
    glVertex2i (20, 50);
  glEnd ();
  glFlush ();
}
void main (int argc, char** argv){
  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition (0, 0);
  glutInitWindowSize (800, 600);
  glutCreateWindow ("The OpenGL example");

  init ();
  glutDisplayFunc (createLine);
  glutMainLoop ();
}
```

And we need to force the system to process all these functions. This we do with the other function glFlush as shown here. So, this should be again the last line of our picture generation procedure which indicates that all these functions that we have used must be processed one after another.

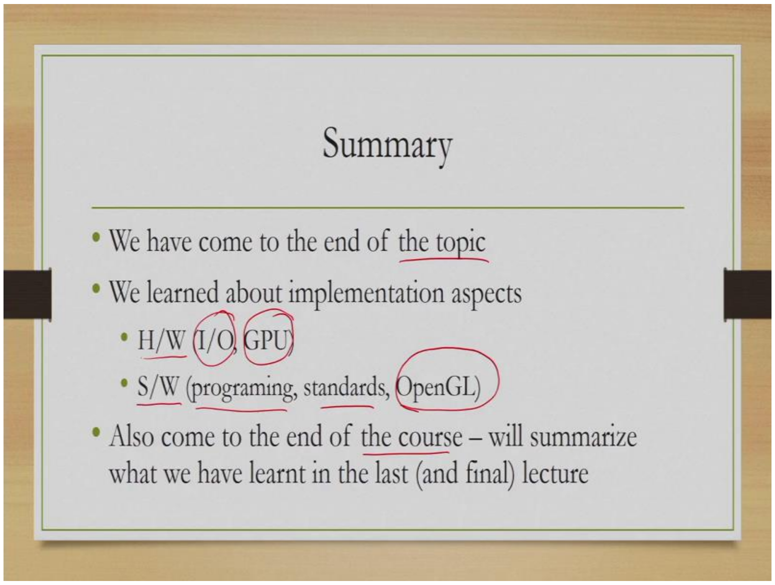
(Refer Slide Time: 37:47)



End Note

So, that is how we can create a program using OpenGL. So, in our example we have used OpenGL library in the setting of C language and we have also seen that only OpenGL library is not sufficient, we need to some auxiliary libraries, here we have used GLUT as well as GLU auxiliary libraries, GLUT stands for GL Utility Toolkit which allows us to create the window which is a display dependent operation and GLU allows us to perform other complex tasks which are not there in core OpenGL library.

(Refer Slide Time: 38:44)

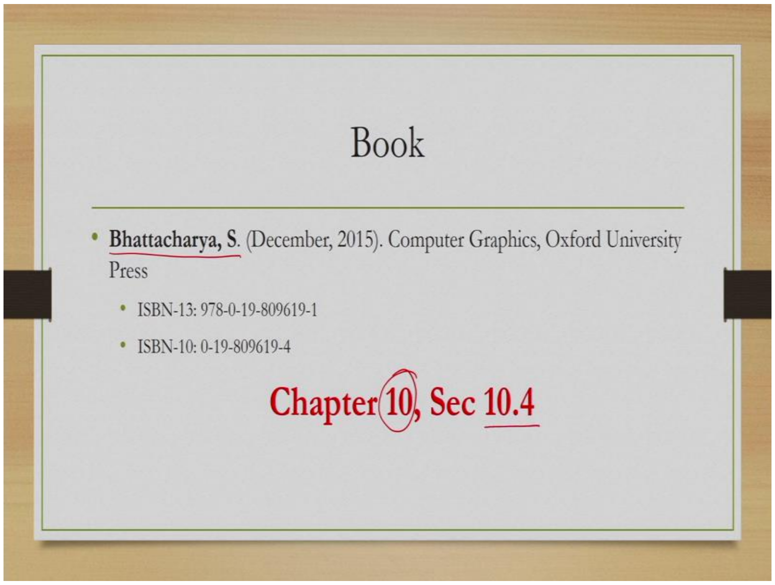


Summary

- We have come to the end of the topic
- We learned about implementation aspects
  - H/W (I/O, GPU)
  - S/W (programing, standards, OpenGL)
- Also come to the end of the course – will summarize what we have learnt in the last (and final) lecture

So, with this we have come to the end of the topic. So, we have learned various things, the graphics hardware including the input output and GPU, also we started with a generic architecture of a graphic system and then learned about various IO and GPU and today we learned about the graphics software, how the softwares are created, different standards and an example program using OpenGL, OpenGL can be used to write any graphics program. Now, with this lecture we have come to the end of the course. So, in the next lecture we will summarize what we have learnt in this course so far.

(Refer Slide Time: 39:44)



Book

- **Bhattacharya, S.** (December, 2015). Computer Graphics, Oxford University Press
  - ISBN-13: 978-0-19-809619-1
  - ISBN-10: 0-19-809619-4

Chapter 10, Sec 10.4

Whatever I discussed today can we found in this book, you can go through chapter 10, section 10.4 to learn on Graphic Software including the OpenGL example. So, in the last lecture we will summarize our learning so far, so we will see you in the concluding lecture, till then thank you and goodbye.