African Virtual University

Applied Computer Science: CSI 4103

COMPUTER GRAPHICS

Dr. Mercy Mbise

Foreword

The African Virtual University (AVU) is proud to participate in increasing access to education in African countries through the production of quality learning materials. We are also proud to contribute to global knowledge as our Open Educational Resources are mostly accessed from outside the African continent.

This module was developed as part of a diploma and degree program in Applied Computer Science, in collaboration with 18 African partner institutions from 16 countries. A total of 156 modules were developed or translated to ensure availability in English, French and Portuguese. These modules have also been made available as open education resources (OER) on oer.avu.org.

On behalf of the African Virtual University and our patron, our partner institutions, the African Development Bank, I invite you to use this module in your institution, for your own education, to share it as widely as possible and to participate actively in the AVU communities of practice of your interest. We are committed to be on the frontline of developing and sharing Open Educational Resources.

The African Virtual University (AVU) is a Pan African Intergovernmental Organization established by charter with the mandate of significantly increasing access to quality higher education and training through the innovative use of information communication technologies. A Charter, establishing the AVU as an Intergovernmental Organization, has been signed so far by nineteen (19) African Governments - Kenya, Senegal, Mauritania, Mali, Cote d'Ivoire, Tanzania, Mozambique, Democratic Republic of Congo, Benin, Ghana, Republic of Guinea, Burkina Faso, Niger, South Sudan, Sudan, The Gambia, Guinea-Bissau, Ethiopia and Cape Verde.

The following institutions participated in the Applied Computer Science Program: (1) Université d'Abomey Calavi in Benin; (2) Université de Ougagadougou in Burkina Faso; (3) Université Lumière de Bujumbura in Burundi; (4) Université de Douala in Cameroon; (5) Université de Nouakchott in Mauritania; (6) Université Gaston Berger in Senegal; (7) Université des Sciences, des Techniques et Technologies de Bamako in Mali (8) Ghana Institute of Management and Public Administration; (9) Kwame Nkrumah University of Science and Technology in Ghana; (10) Kenyatta University in Kenya; (11) Egerton University in Kenya; (12) Addis Ababa University in Ethiopia (13) University of Rwanda; (14) University of Dar es Salaam in Tanzania; (15) Universite Abdou Moumouni de Niamey in Niger; (16) Université Cheikh Anta Diop in Senegal; (17) Universidade Pedagógica in Mozambique; and (18) The University of the Gambia in The Gambia.

Bakary Diallo

The Rector

African Virtual University

Production Credits

Author

Mercy Mbise

Peer Reviewer

Robert Oboko

AVU - Academic Coordination

Dr. Marilena Cabral

Overall Coordinator Applied Computer Science Program

Prof Tim Mwololo Waema

Module Coordinator

Florence Tushabe

Instructional Designers

Elizabeth Mbasu

Benta Ochola

Diana Tuel

Media Team

Sidney McGregor

Barry Savala

Edwin Kiprono

Kelvin Muriithi

Victor Oluoch Otieno

Michal Abigael Koyier

Mercy Tabi Ojwang

Josiah Mutsogu

Kefa Murimi

Gerisson Mulongo

Copyright Notice

Supported By

AVU Multinational Project II funded by the African Development Bank.

Table of Contents

Course Overview

Welcome to Computer Graphics

This course is about learning how to create and manipulate two and three dimensional images using graphic techniques and methods. Computer graphics knowledge enables one to create and manipulate graphics in application areas such as digital art and graphics design. At the end of the course you will be able to understand and apply the theory of computer graphics and in particular the mathematical techniques and the algorithms that underline most of modern 3D graphics systems. In general you will be able to write your own 3D graphics programs.

Prerequisites

- Object-oriented Programming
- Advanced Mathematics
- Basic Image Properties and Processing

Materials

The materials required to complete this course are:

- An application program interface software for defining 2 and 3 dimensional graphic images, such as OpenGL.
- A program that translates source code (written in a programming language, such as C++) into computer executable code. Example is the C++ Compiler.
- A specialized electronic circuit for manipulating and altering memory to accelerate the creation of images. This is called a Graphic Processor Unit (GPU) or Visual Processing Unit(VPU).

Course Goals

Upon completion of this course the learner should be able to:

1. State and define the basic concepts and elements of computer graphics.
2. Identify and define various picture transformation techniques.
3. Employ transformation techniques on 2D and 3D graphics.
4. Create and manipulate simple graphics using OpenGL.

Units

Unit 0: Pre-Assessment

This unit introduces pre-requisite basics that are necessary in order to undertake the Computer Graphics Module. In this unit, topics in advanced Mathematics and object-oriented programming, which are essential for computer graphics, will be introduced.

Unit 1: Basics and Elements of Computer Graphics

This unit introduces the basic concepts and applications of computer graphics. Elements of computer graphics, and how colour is represented in computers will be covered in this unit.

Unit 2: Picture Transformations

In this unit, image manipulation techniques, such as translation, transformation and rotation will be introduced. These are necessary so that an image or a picture is not redrawn but rather manipulated to produce a different version, size or look of it.

Unit 3: Creating Graphics with OpenGL

This unit introduces the OpenGL Language for graphics programming. The language will be the tool for understanding 2D and 3D graphics programming. Basic concepts will be introduced with some simple demo codes.

Unit 4: 3D Graphics

This unit introduces students to 3D graphics. Students will be introduced to the concepts of 3D dimension and will learn how to construct and manipulate 3D graphics.

Assessment

Formative assessments, used to check learner progress, are included in each unit.

Summative assessments, such as final tests and assignments, are provided at the end of each module and cover knowledge and skills from the entire module.

Summative assessments are administered at the discretion of the institution offering the course. The suggested assessment plan is as follows:

1	Tests	20%
2	Assignments	20%
3	LAB	20%
4	Final Exam	40%
Total		100%

Schedule

Unit	Activities	Time
Unit 0: Pre-assessment	0.1 Advanced Mathematics - Trigonometry - Polar coordinates - 3-D coordinate system - Points - Vectors - Matrices 0.2 Object-oriented programming - The “object” concept - Object-oriented paradigm - Abstraction - Polymorphism - Inheritance - Encapsulation	10hrs

Unit 1: Basics and Elements of Computer Graphics	1.1 Basic Image properties and processing - Image basics - Colour basics - Computer vision basics 1.2 What is computer graphics? 1.3 Computer graphics applications 1.4 Polylines 1.5 Text 1.6 Filled regions 1.7 Color representation 1.8 Input and output devices (CRT)	20hrs
Unit 2: Picture Transformations	2.1 What is transformation? 2.2 Basic transformation 2.3 Matrix representation of points 2.4 Translation 2.5 Rotation 2.6 Scaling 2.7 2D Viewing pipeline	30hrs
Unit 3: Creating Graphics with OpenGL	3.1 What is OpenGL? 3.2 OpenGL Data types 3.3 Graphics initialization 3.4 Creating graphics window 3.5 Basic graphics primitive routines 3.6 Interaction with Mouse and keyboard 3.7 Viewports 3.8 Window to viewport mapping 3.9 Clipping 3.10 Zooming	30hrs
Unit 4: 3D Graphics	4.1 Logical classification of input devices 4.2 Interactive input techniques 4.3 3D transformations 4.4 3D viewing pipeline 4.5 Hidden line and hidden surface removal	30hrs
	Total	120hrs

Readings and Other Resources

The readings and other resources in this course are:

Unit 0

Required readings and other resources:

- Essential Mathematics for Computer Graphics fast, Vince John, 2001, Springer
- 4 Major Principles of Object-oriented Programming, Raymond Wallen (accecible from http://codebetter.com/raymondlewallen/2005/07/19/4-major-principles-of-object-oriented-programming/)

Unit 1

Required readings and other resources:

- Fundamentals of Image Processing, Ian T. Young, Jan J. Gerbrands, Lucas J. van Vliet,1998, Delft University of Technology (downloadable free from http://www.ebook3000.com/Fundamentals-of-Image-Processing_60576.html)
- Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya®. New York: Springer Science, Shalini Govil-Pai (2004).
- Essential Mathematics for Computer Graphics fast, Vince John, 2001, Springer

Optional readings and other resources:

- http://nmita.iowa.uiowa.edu/instr_co.htm

Unit 2

Required readings and other resources:

- Shalini Govil-Pai (2004). Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya®. New York: Springer Science
- Essential Mathematics for Computer Graphics fast, Vince John, 2001

Optional readings and other resources:

- http://www.pling.org.uk/cs/cgv.html

Unit 3

Required readings and other resources:

- Francis S Hill, Stephen M Kelley (2006) Computer Graphics Using OpenGL (3rd Edition), Prentice Hall

Optional readings and other resources:

- www.glprogramming.com/blue/ch01.html

Unit 4

Required readings and other resources:

- Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya®. New York: Springer Science, Shalini Govil-Pai (2004).
- Essential Mathematics for Computer Graphics fast, Vince John, 2001, Springer

Unit 0. Pre-Assessment

The purpose of this unit is to determine a student's grasp of knowledge related to this course. It introduces and tests a student on pre-requisite basics that are necessary to undertake the Computer Graphics Module. Some topics in advanced Mathematics and object-oriented programming will be introduced.

Unit Objectives

Upon completion of this unit you should be able to:

- Illustrate basic trigonometric functions of an angle (sine, cosine and tangent)
- Define and illustrate a point, a vector and a matrix
- Illustrate the Cartesian (x,y) and Polar (r,θ) coordinate systems
- Illustrate the 3D Cartesian coordinate system
- Define the OOP terms
- List and describe/define the properties of an object (abstraction, polymorphism, inheritance, encapsulation)

Key Terms

OOP: object-oriented programming - a programming paradigm that represents the concept of objects that have data fields and associated procedures known as methods point, vector and matrix

Cartesian: a coordinate system that uniquely specifies each point in a plane by a pair of numerical coordinates in the horizontal and vertical directions

2D: two-dimensional space

3D: three-dimensional space

Matrix: a rectangular array of numbers, symbols, or expressions, arranged in rows and columns

Unit Assessment

Assessment Title: Unit 0 Assessment

Review Questions

This is a pre-assessment unit; marks awarded in this unit do not count for the assessment of the overall module.

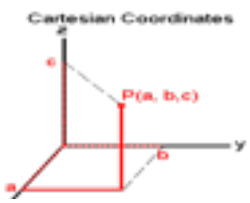
Assessment Items:

Advanced Mathematics

1. Using a diagram, illustrate the basic trigonometric functions of an angle and their mathematical expressions.
2. Differentiate between a point, a vector and a matrix.
3. The position of a point is defined by specifying the distance (radius) from a fixed point called as …….
4. Differentiate between Cartesian and Polar coordinate systems.
5. Using a diagram, illustrate the 3D Cartesian coordinates.
6. A point in Cartesian coordinate system can be defined by specifying two numbers, called as …….. and the …… of that point.

Answers

1. Sineθ=BC/BA Cosθ=AC/AB Tanθ=Sinθ/Cosθ
2. Points are always related to an origin, while vectors can are independent of any origin and matrix is a 2-dimensional grid of numbers
3. origin
4. The Cartesian coordinate system is a two-dimensional coordinate system using a rectilinear grid while the polar coordinate system is a two-dimensional coordinate system using a polar grid:
5. In a 3D Cartesian coordinate system, a point P is referred to by three real numbers (coordinates), indicating the positions of the perpendicular projections from the point to three fixed, perpendicular, graduated lines, called the axes which intersect at the origin.



6. x coordinate and y coordinate

Object-oriented programming

7. Define the following terms as applied in object-oriented paradigm
   - 7.1. Abstraction
   - 7.2. Polymorphism
   - 7.3. Inheritance
   - 7.4. Encapsulation
8. Give an example of any real world application in which abstraction can be used.
9. Write two advantages of inheritance as far as OOP is concerned.
10. Differentiate between abstraction and encapsulation.
11. Read the following piece of code and then answer the following questions:

```
public abstract class Vehicle
{
    public abstract int Wheels;
}
public class Bicycle : Vehicle
{
    public override int Wheels()
    {
        return 2;
    }
}
public class Car : Vehicle
{
    public override int Wheels()
    {
        return 4;
    }
}

public class Truck : Vehicle
{
    public override int Wheels()
    {
        return 18;
    }
}
```

11.1 Name the base class and sub-classes.

11.2 What item in the base class is generic to all the other sub-classes? Explain why, based on your understanding on polymorphism.

Answers

7.1: Abstraction means working with something we know how to use without knowing how it works internally. i.e hiding hides all but the relevant data about an object in order to reduce complexity and increase efficiency.

7.2 Encapsulation Is the act of an object to provide its users only with the essential information for manipulation, without the internal details.

7.3 Inheritance is the ability of a class to "inherit" (behavior or characteristics) of another, more general class.

7.4 Polymorphism allows treating objects of a derived class as objects of its base class

A good example of abstraction is a television set. We don't need to know the inner workings of a TV, in order to use it. All we need is a remote control with a small set of buttons (the interface of the remote) and we will be able to watch TV.

9.1 Reusability

9.2 Extensibility

9.3 Overriding

10. Encapsulation is used for hide the code and data in a single unit to protect the data from the outside the world. Class is the best example of encapsulation WHILE Abstraction refers to showing only the necessary details to the intended user.

11.1 Base Class: Vehicle and Subclass: Truck, Car and Bicycle

11.2 The method wheel is general to all sub classes which was used but overriding in order to change their original behavior inherited from the base class.

Grading Scheme

This unit will not be graded.

Unit Readings and Other Resources

The readings in this unit are to be found at the course-level section "Readings and Other Resources".

Unit 1. Basics and Elements of Computer Graphics

Unit Introduction

This unit introduces the basic concepts and applications of computer graphics. The basics of digital images will be addressed, as well as how colour is represented in computers. The unit will also address elements of computer graphics, including examples of where computer graphics can be applied. The unit is essential in building the foundation knowledge to computer graphics e.g. image and video processing

Unit Objectives

Upon completion of this unit you should be able to:

- Define the following terms: computer graphics, digital image
- List the properties of a digital image
- Describe how colour is represented in a computer
- List examples of applications of computer graphics
- Illustrate the elements of a picture, such as polylines, text and filled regions
- List input and output devices

Key Terms

Pixel: The smallest unit of a digital image

Resolution: The number of pixels in an image

Graphics: The use of diagrams in calculation and design

Intensity: The measurable amount of brightness

Raster: A rectangular pattern of parallel lines

Bitmap: A representation in which each item corresponds to one or more bits of information

Vector: A quantity having direction and magnitude, especially as determining the position of one point in space relative to another.

Learning Activities

Activity 1.1 - Basic Image Properties

Introduction

In this activity, the definition of image and object as applied in Computer Graphics is presented.

Activity Details

Real world objects, such as cars, houses, trees, etc, are generally in three-dimensional world coordinates. These objects can be represented in a computer using images. An image, or simply a picture, can be mathematically represented using a two-dimensional array of data with intensity or a color value at each element of the array. In computer graphics, these values are a collection of discrete (digital) picture elements called pixels.

A pixel is the smallest element of a picture that can be represented on the screen of a device like a computer. Figure 1.1(a) shows relationship between a two-dimensional (X and Y) image and pixels within the image.

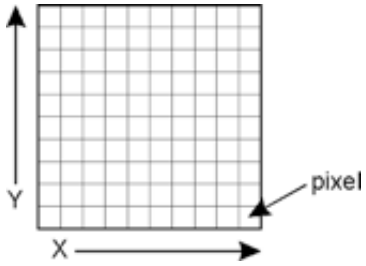


Figure 1.1(a): pixels in an image

The number of pixels in an image determines its resolution. Typical resolutions range from 320 by 200 pixels to 2000 by 1500 pixels. The intensity of each pixel is described by a number. For example, for a black-and-white image, the pixels can be expressed between 0.0 (black) and 1.0 (white). Since computers store data in binary form, these values are stored as integers between 0 (black) and 255 (white). Figure 1.1(b) shows an image in colour as well as in black and white.

Figure 1.1 (b): Black and White vs Colour Image

In colour images on the other hand, the pixels are described by three numbers representing the intensity of the three basic (primary) colours: red, green and blue. For example, pure red is (255,0,0); pure green is (0,255,0), etc. A combination of these values of colours results in other (secondary) colours, as depicted in Figure 1.1 (c).

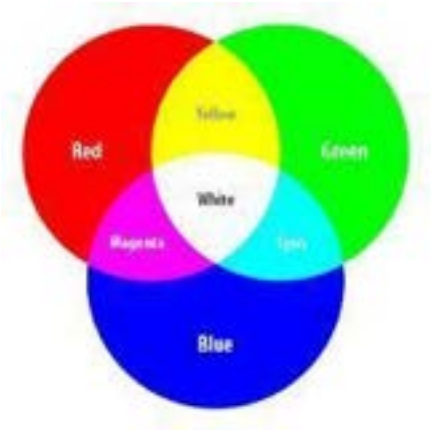


Figure 1.1(c): Colour Space

Conclusion

Objects are represented in a computer using two-dimensional images. Images consist of small, discrete values called pixels. Pixels are represented in a computer using integer values between 0 and 255. Colour pixels are represented using three numbers representing the intensity of red, green and blue.

Activity 1.2 - Computer Graphics and its Applications

Introduction

Computer Graphics involves the ways in which images can be displayed, manipulated and stored using a computer. Computer graphics provides the software and hardware techniques or methods for generating images. In this activity, the definition of Computer Graphics will be presented, together with examples of where computer graphics is applied.

Activity Details

Definition

Computer graphics is an art of drawing pictures, lines, charts, etc, using computers with the help of programming. Computer graphics are made up of number of pixels. A pixel is the smallest graphical picture or unit represented on the computer screen.

Graphics in a computer can be either interactive or non-interactive.

Interactive computer graphics: It is the computer graphics in which a user can interact with the image on the computer screen. There exists a two-way communication between the user and the image. The image is totally under the control of a user. Example: Playing computer games in a computer in which a user controls the image completely.

Non-interactive computer graphics: It is the computer graphics in which a user does not have any kind of control over the image. The image is merely the product of static stored program and will work according to the instructions given in the program linearly. The image is totally under the control of program instructions and not under the user. Example: screen savers.

Computer graphics can be classified into two categories: **Raster** or **Bitmap** graphics and **Vector** or **Object-oriented** graphics.

1. Raster (Bitmap) Graphics

- These are pixel based graphics and the pixels can be modified individually.
- The images are easy to edit in memory and display on TV monitors due to the arrangement of the pixels in a rectangular array.
- The image size is determined on the basis of image resolution.
- These images cannot be scaled easily; resizing do not work very well and can significantly distort the image.
- Bitmap graphics are used for general purpose images and in particular photographs.

Figure 1.2(a) illustrates the concept of bitmap images.

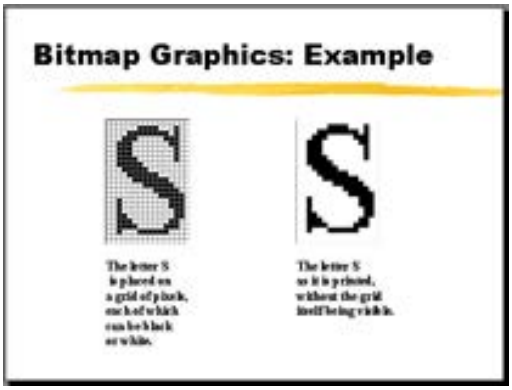


Figure 1.2(a): Bitmap Image

2. Vector (Object-oriented) Graphics

- These graphics are mathematically based images.
- Vector based images have smooth edges and therefore used to store images composed of lines, circles and polygons.
- These images can easily be re-scaled and rotated.
- They can not easily accommodate complex images such as photographs where colour information varies from pixel to pixel.
- Vector graphics are well suited for graphs, e.g. in spreadsheets and for scalable fonts, e.g. postscript fonts

Figure 1.2(b) illustrates the concept of vector images.

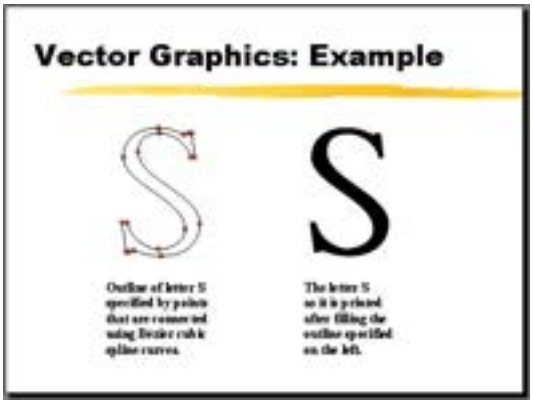


Figure 1.2(b): Vector Image

Figure 1.2(c) illustrates the difference between a vector and a bitmap image.

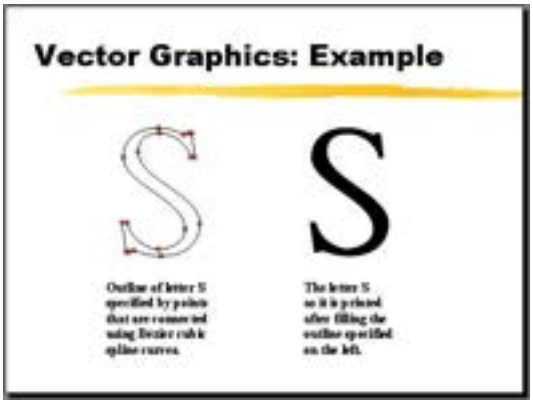


Figure 1.2 (c): Vector vs Bitmap Image

Applications of Computer Graphics

Computer graphics can be applied in various areas. Examples are such as follows:

Computer-Aided Design:

In engineering and architectural systems, the products are modeled using computer graphics commonly referred as CAD (Computer Aided Design). In many design applications like automobiles, aircraft, spacecraft, etc., objects are modeled in a wireframe outline that helps the designer to observe the overall shape and internal features of the objects.

Computer Art:

A variety of computer methods are available for artists for designing and specifying motions of an object. The object can be painted electronically on a graphic tablet using stylus with different brush strokes, brush widths and colors. The artists can also use combination of 3D modeling packages, texture mapping, drawing programs and CAD software to paint and visualize any object.

Entertainment:

Computer graphics methods are widely used in making motion pictures, music videos and television shows. Graphics objects can be combined with live actions or can be used with image processing techniques to transform one object to another.

Education and training:

Computer graphics can make better the understanding of the functioning of a system. In physical systems, biological systems, population trends, etc., models make it easier to understand. In some training systems, graphical models with simulations help a trainee to train in virtual reality environment. For example, practice session or training of ship captains, aircraft pilots, air traffic control personnel.

Image processing:

Image processing provides techniques to modify or interpret existing images. One can improve picture quality through image processing techniques. For instance, in medical applications, image processing techniques can be applied for image enhancements and is been widely used for CT (Computer X-ray Tomography) and PET (Position Emission Tomography) images.

Graphical User Interface:

GUI is commonly used to make a software package more interactive. There are multiple window systems, icons, menus, which allow a computer setup to be utilized more efficiently.

Conclusion

Computer Graphics involves ways in which images can be displayed, manipulated and stored using computers. Computer graphics images can be categorised into raster graphics, which as pixel-based graphics, and vector graphics, which are mathematically represented. Computer graphics is applicable in various areas such as computer-aided design, computer art, entertainment, as well as in education and training.

Activity 1.3 - Computer Graphics Elements (Primitives)

Introduction

This activity presents how output primitives of graphics images appear. These primitives are polylines, text, and filled regions. All other graphic elements are built from these primitives.

Activity Details

Polylines

A polyline is a connected sequence of straight lines. To the eye, a polyline can appear as a smooth curve. Simple polyline attributes are colour and thickness. The simplest polyline is a single straight line segment. A line segment is specified by its two endpoints, such as (x1, y1) and (x2, y2). When there are several lines in a polyline, each one is called an edge, and two adjacent lines meet at a vertex.

The edges of a polyline can cross one another but a polyline does not have to be closed. A polygon has its first and last points connected by an edge. If no two edges cross, the polygon is called a simple polygon. An example of a polyline is shown in figure 1.3(a), and a polygon is shown in figure 1.3(b).

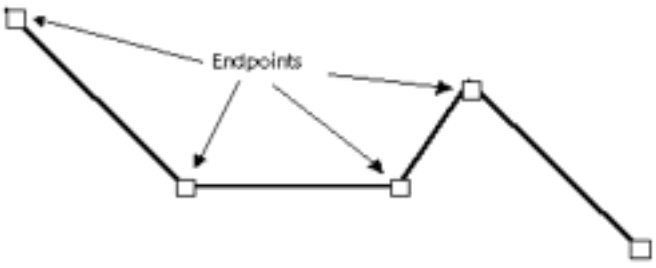


Figure 1.3 (a) Polyline

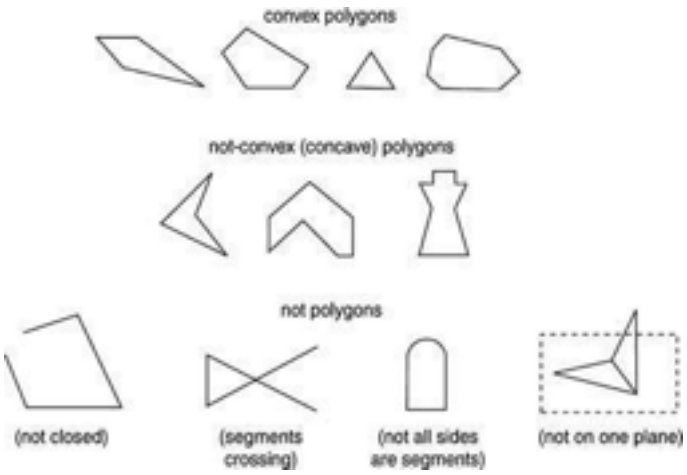


Figure 1.3(b) Polygon

Text

Some graphics devices have two distinct display modes: text mode and graphics mode. In text mode, text is generated using a built-in character generator. Text in graphics mode is drawn. Text attributes are such as colour, size, font, spacing and orientation.

Filled Regions

A filled region is a shape filled with some colour or pattern. An example is a filled polygon as shown in figure 1.3(c).

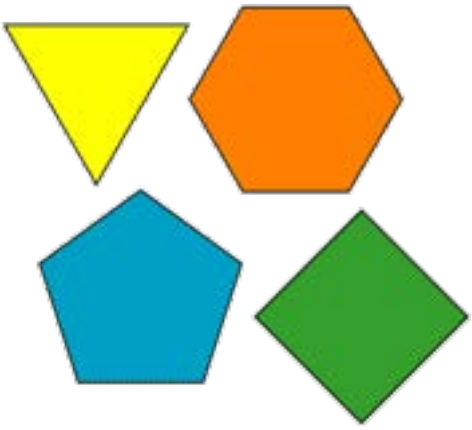


Figure 1.3(c) Filled Regions

Conclusion

In this activity, the basic graphics primitives: polylines, text and filled regions were presented. They are characterized by attributes such as colour, size, font and thickness. These primitives are used to create other graphics.

Unit Summary

This unit presented the basics of computer graphics, including objects, basic image properties and pixels. The unit has presented ways in which images can be displayed, manipulated and stored using computers. It has introduced the raster graphics, which as pixel-based graphics, and vector graphics, which are mathematically represented. The elements of computer graphics, i.e. polylines, text and filled regions, were addressed. Lastly, applications of computer graphics in education, entertainment, computer art, etc were presented.

Unit Assessment

Assessment Title: UNIT 1 Assessment

Instructions: Answer the following questions

1. Define an image. (1 mark)
2. Define an object. (1 mark)
3. Define a pixel. (1 mark)
4. What is an image's aspect ratio? (1 mark)
5. What is image resolution? (1 mark)
6. How is an image represented Mathematically? (2 marks)
7. Briefly describe two basic types of Graphics file formats. (2 marks)
8. List two categories of images in computer graphics. (2 marks)
9. Which graphic system is better for photorealistic images? (1 mark)
10. In which graphic system are images easily scalable? (1 mark)
11. Frame buffers are described as having a certain depth. What does the depth refer to? (2 marks)
12. List four applications of computer graphics. (2 marks)
13. Explain how computer graphics are useful in entertainment industry and in education. (3 marks)
14. List the basic graphics primitives. (2 marks)
15. Explain the difference between a polygon and a filled region. (2 marks)
16. What attributes characterize each of the primitives? (2 marks)
17. In reference to polyline; define the terms edge and vertex. (2 marks)

Answers

These are solutions to the above questions

1. Image is a representation of an object in a 2 dimension.
2. An object is generally a three-dimensional world coordinates
3. A pixel is the smallest element of a picture that can be represented on the screen of a device like a computer.
4. The ratio of its width to its height, measured in unit length or number of pixels
5. Image resolution is the number of pixels in an image can be represented
6. Mathematically using a two-dimensional array of data with intensity or a color value at each element of the array
7. Paint packages such as Paint or Paint Shop Pro produce bitmap images. Such packages record the image on the screen as an array of dots of various colors or grey-scale intensities. Having created a line, for example, it is not possible with Paint packages to then extend the line or move it on the page. All you can do is erase it and re-paint it.

   AND

   Drawing packages such as Micrografx Designer, Microsoft Draw and CorelDRAW use these. Images stored in these formats can be resized on screen without distortion and can take advantage of the high resolution capabilities of laser printers. Common vector-based formats are Encapsulated PostScript (EPS), Computer Graphics Metafile (CGM), Micrografx Drawing (DRW), AutoCAD (DXF), Hewlett Packard Graphics Language
8. Raster and Vector Graphics images
9. Bitmap
10. Vector
11. The number of bits used to store the color for each pixel
12. Computer-Aided design, computer art, entertainment, image processing, education and training
13. Can be used to transform one object to another to bring easier understanding of a concept or action
14. Basic graphics primitives are polylines, text, and filled regions
15. Polygon is a connected sequence of straight lines while Filled region is a shape or polygon filled with color

16. Polylines are characterized by color and thickness; Text is characterized by colour, size, font, spacing and orientation; Filled region is characterized by color, thickness and size

17. An edge is side line in a polyline and vertex is a point where two adjacent edges meet.

Unit 2. Picture Transformations

Unit Introduction

Transformations are one of the fundamental operations performed in computer graphics. In computer graphics, images are not only created, but also transformed to make them appear different from what they look like. This unit presents definitions and basic picture transformation techniques for object attributes, such as changing their position, size and orientation as viewed on display devices. The unit also presents the 2D viewing pipeline where 2D world coordinate system images are mapped to device coordinates.

Unit Objectives

Upon completion of this unit you should be able to:

- Define transformation.
- Identify the types of transformation operations.
- Illustrate representation of a point using matrix.
- Perform translation, scaling and rotation on images.

Key Terms

World coordinates: Dimensional coordinates that define the location of objects in the real world.

Viewpoint (Camera) coordinates: The coordinates are based upon the viewpoint of the observer, and changes as they change their view.

Screen coordinates: Physical coordinates of the pixels on the computer screen, based on current screen resolution.

Transform: Changing some graphics into something else by applying rules

Translation: Shifting of a point to some other place, whose distance with regard to the present point is known

Rotation: Rotation of a point about an axis

Scaling: Increasing or decreasing the size of a picture in one or either directions

Learning Activities

Activity 2.1 – Definitions and Transformation Basics

Introduction

When objects defined in one coordinate system are required to be observed on another coordinate system, transformation techniques are usually used. Transformation is performed such as to shift a picture from one place to some other place on a screen, or to increase or decrease its size. This activity will present the definition of transformation, matrix representation of points, and the basics of transformation.

Activity Details

What is Transformation?

Transformation is a process which is used to obtain a different kind of picture from an existing one by manipulating the attributes of the original picture. These attributes are position, size, and orientation of the picture.

Matrix Representation of Points

A picture can be thought of as being a combination of points. The idea is to identify the points which form a picture that is being drawn or displayed. A screen is also thought to be a made up of a number of pixels, each pixel corresponding to a point. Those pixels, which form a part of the picture being drawn are made to light up so that the picture is visible on the screen. Based on the Cartesian coordinate system, a point is represented by two values as **(x, y). x** represents the distance of the point from the origin in the horizontal direction and y in the vertical direction. In the context of graphics, a point is represented as a 3 valued entity [x, y, 1]. **X** and **Y** are the coordinates, and 1 is a value which is used for representation. Figure 2.1(a) shows a point on a Cartesian coordinate system at position (4, 2).

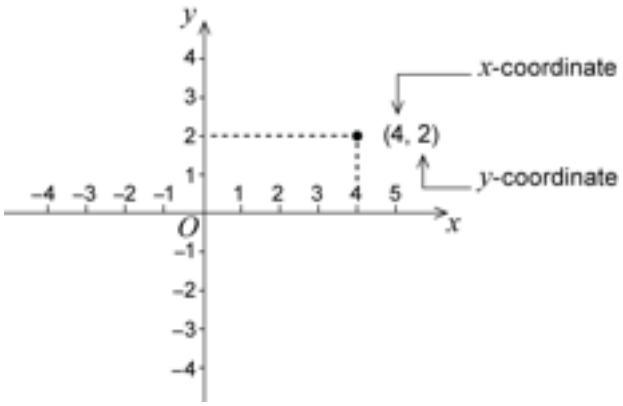


Figure 2.1(a) A point on Cartesian coordinates

Basics of Transformation

Manipulation of images on a screen involves two kinds of processes. The first one deals with changing the appearance of the picture by modifying its colour and surface texture, as well as changing its appearance by changing the lightning model that the computer uses to make it visible.

The second process is geometric, in which the relative or absolute location of the picture in the coordinate system is changed. The picture in this case may be moved around the screen, or appear to become larger or smaller. For example, a picture can be moved from the centre to the top left hand corner of a screen. The picture can also be increased or decreased to twice or half its size, respectively. The picture can also be turned/rotated 45 or 90 degrees.

The three basic transformations are:

1. Translation
2. Rotation
3. Scaling

These will be discussed in details in the next activity.

Conclusion

Pictures on computer screens can be translated to different positions on the screen, or resized, or rotated to appear differently from the original picture. This activity has presented the definition of transformation, and introduced the types of picture transformations.

Activity 2.2 – Types of Transformations

Introduction

As introduced in the previous activity, there are three basic transformations: translation, rotation and scaling. These transformations are essentially mathematical operations which are performed on points and hence pictures, since a picture can be viewed as a collection of points.

Activity Details

Translation

Translation refers to the shifting of a point to some other place, whose distance with regard to the present point is known. This process shifts the position of a point from one set of coordinates to another.

Consider a point P(x1, y1) to be translated to another point Q(x2, y2). If the point values of (x2, y2) are known, the point can directly be shifted to Q by displaying the pixel at (x2, y2). If the distance by which the point is to be shifted is known, for instance Tx and Ty along the x and y axes respectively, then the coordinates can be derived by x2 =x1 +Tx and y2 = y1+ Ty.

Example: if a triangle with coordinates A(20,10), B(30,100) and C(40,70) is to be shifted by 20 units along the x axis and 10 units along the y axis, then the new triangle will be at A1(20+20, 10+10) B1 (30+20, 10+10) C1(40+20, 70+10).

In matrix form, [x2 y2 1] = [x1 y1 1] *

Rotation

Rotation is the rotation of a point about an axis. The axis can be any of the coordinates or any other specified line. Suppose a point (x1 y1) is to be rotated clockwise through an angle θ about the origin of the coordinate system. Mathematically the new coordinates of the point (x2 y2) can be expressed as:

$$x2 = x1\cos\theta + y1\sin\theta$$

$$y2 = x1\sin\theta - y1\cos\theta$$

These equations are applicable only if the rotation is about the origin. The matrix for [x2 y2 1] = [x1 y1 1] *

Scaling

Scaling is the increasing or decreasing the size of a picture in one or either directions. The size of the picture is changed by increasing or decreasing the distance between the end points of the picture and changing the intermediate points as per requirements.

Suppose a point (x1 y1) is to be scaled by a factor sx along the x direction and by a factor sy along the y direction, the new coordinates become:

x2 = x1 * sx

y2 = y1 * sy

Scaling a point physically means moving the point away but does not magnify the point. However, when a picture is scaled, each of the points are scaled differently and hence the dimensions of the picture changes.

The translation, rotation and scaling are summarized in Figure 2.2(a).

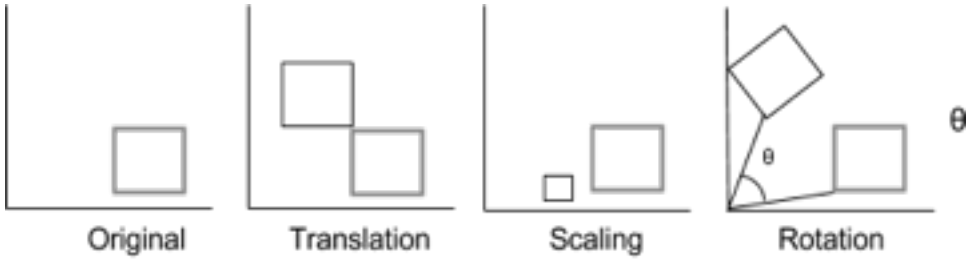


Figure 2.2(a) Types of transformations

Conclusion

This activity has presented the types of transformations, i.e. translation, scaling and rotation. In translation, a picture's position is altered by altering its coordinates. In scaling, a picture's size is altered, i.e. increased or decreased, while in rotation, a picture is rotated about an axis through an angle, clockwise or anticlockwise. These operations are necessary in computer graphics such that pictures can be manipulated and transformed in terms of their size, position and orientation, to give various effects of the pictures.

Activity 2.3 – 2D Viewing Pipeline

Introduction

In many cases, the size of display devices is generally limited in terms of physical dimension as well as resolution. When displaying pictures on devices, their dimensions may be limited by the size of the display device. In such cases, the pictures need to be "fitted" on the display device, and undergoes a series of transformations, which are termed viewing pipeline, so that it can be displayed on a physical device.

Activity Details

The term Viewing Pipeline describes a series of transformations, which are passed by geometry data to end up as image data being displayed on a device. The 2D viewing pipeline describes this process for 2D data as shown in Figure 2.3(a).

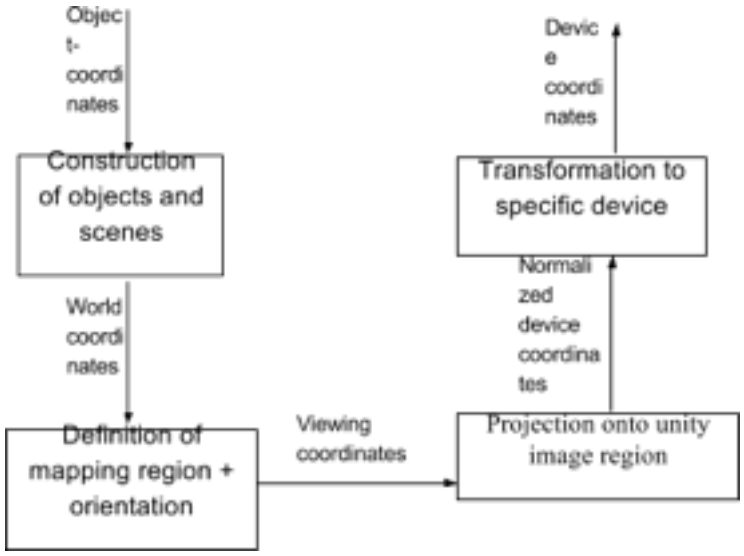


Figure 2.3(a) Viewing Pipeline

- The coordinates in which individual objects (models) are created are called ***model (or object) coordinates.***
- When several objects are assembled into a scene, they are described by ***world coordinates.***
- After transformation into the coordinate system of the camera (viewer) they become ***viewing coordinates.***
- Their projection onto a common plane (window) yields device-independent ***normalized coordinates.***
- Finally, after mapping those normalized coordinates to a specific device, we get ***device coordinates.***

Window Viewport Transformation

A window-viewport transformation describes the mapping of a (rectangular) window in one coordinate system into another (rectangular) window in another coordinate system. This transformation is defined by the section of the original image that is transformed (clipping window), the location of the resulting window (viewport), and how the window is translated, scaled or rotated (c.f. Figure 2.3(b)).

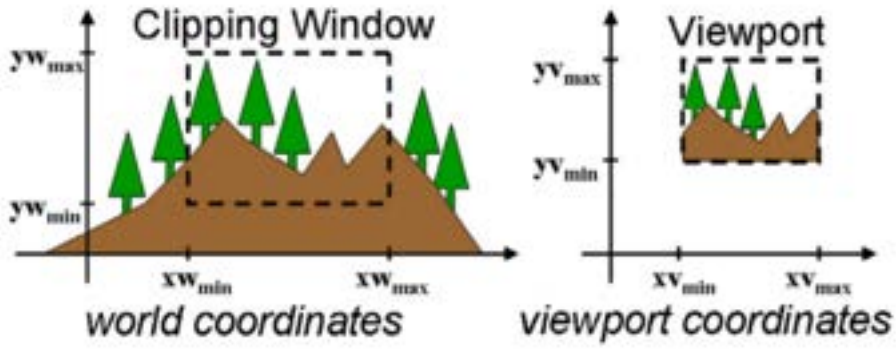


Figure 2.3(b) World to view point coordinates

Conclusion

The viewing pipeline allows the transformation of picture coordinates to a format that can be displayed on a physical display device.

Unit Summary

This unit has presented the fundamentals of transformation and different operations which can be performed on pictures, specifically translation, rotation and scaling. A picture on a computer screen can be translated to different positions on the screen, resized, or rotated to appear differently from the original picture. These operations are necessary in computer graphics such that pictures can be manipulated and transformed in terms of their size, position and orientation, to give various effects. The unit also presented the 2D viewing pipeline where 2D world coordinate system images are mapped to device coordinates.

Unit Assessment

Assessment Title: Unit 2 Assessment

Instructions: Answer the following questions

1. Define transformation. (1 mark)
2. Why is transformation performed? (1 mark)
3. List three ways in which a picture can be transformed. (2 marks)
4. If a point (x, y) is moved to a point which is at a distance of Tx along the x axis, what is its new position? (2 marks)
5. If a point (x, y) is moved to a point which is at a distance of Ty along the y axis, what is its new position? (2 marks)
6. If a point (x, y) is rotated anticlockwise through an angle about the origin, what are its new coordinates? (2 marks)
7. Define the term "Viewing Pipeline". (1 mark)
8. Outline the importance of the viewing pipeline. (2 marks)
9. Differentiate the following terms: window and viewport. (2 marks)
10. Derive the transformation that rotates an object point by an angle θ about the origin, Write the matrix representation for this rotation. (4 marks)
11. Show how you can transform coordinate point P(x,y) in one Cartesian system to the coordinate P'(x',y') in another Cartesian system that is rotated by an angle θ about the origin. (4 marks)

Grading Scheme

Marks are indicated next to each question.

Answers

These are solutions to the above questions

1. Transformation is a process which is used to obtain a different kind of picture from an existing one by manipulating the attributes of the original picture
2. Transformation is done to make an object appear different from the original picture.
3. Translation, Rotation and Scaling
4. (x+Tx, y)
5. (x,y+Ty)
6. (cos θ,sinθ)
7. Viewing Pipeline is a series of transformations, which are passed by geometry data to end up as image data being displayed on a device
8. Viewing Pipeline allows the transformation of picture coordinates to a format that can be displayed on a physical display device.
9. Window is a world-coordinate area selected for display while viewport is an area on a display device to which a window is mapped.
10. 10. $x' = r\cos(\theta + \Phi)$ $y' = r\sin(\theta + \Phi)$

    $x = r\cos\Phi$ and $y = r\sin\theta$ then by cosine and sine rules

    $x' = r\cos\theta\cos\Phi - r\sin\theta\sin\Phi = x\cos\theta - y\sin\theta$

    $y' = r\sin\theta\cos\Phi + r\cos\theta\sin\Phi = x\sin\theta + y\cos\theta$

    $$\begin{matrix} x' & \cos\theta & -\sin\theta & x \\ y' & \sin\theta & \cos\theta & y \end{matrix}$$
11. int x1 = x*Math.cos(θ) – y*Math.sin(θ);

    int y1 = x*Math.sin(θ) + y*Math.cos(θ);

    Pixel pixel = getPixel(x,y);

    setPixel(pixel,x1,y1);

Unit 3. Creating Graphics with OpenGL

Unit Introduction

Computer graphics is mostly mastered by practicing; such as by writing and testing programs that produce a variety of pictures. An environment that allows one to write and execute programs is required. The environment should generally include hardware for display of pictures, and software tools that written programs can use to perform the actual drawing of pictures. This unit presents the Application Programming Interface (API) for use in producing graphics.

Unit Objectives

Upon completion of this unit you should be able to:

- Define OpenGL.
- Describe the fundamental elements of OpenGL.
- Write programs for making simple graphics in OpenGL
- Illustrate object's viewports and perform clipping on graphics.

Key Terms

Computer program: a collection of instructions that performs a specific task when executed by a computer

OpenGL: Application Programming Interface for creating 2D and 3D graphic images

Clipping: a method to selectively enable or disable rendering operations within a defined region of interest

Graphics primitives: A basic non-divisible graphical element for input or output within a computer-graphics system

Vertex: data structure that describes certain attributes, like the position of a point in 2D or 3D space

Learning Activity 3.1 – OpenGL: Definition and Fundamentals

Introduction

In this activity, the definition and fundamentals of OpenGL as an API will be presented.

Activity Details

Definition

OpenGL is an application programming interface (API) for creating 2D and 3D graphic images. OpenGL specifies a set of commands in which each command directs a drawing action or causes special effects. OpenGL is independent of windowing characteristics of each operating system.

OpenGL Fundamentals

OpenGL allows one to draw graphics primitives (lines, polygons, points, etc). Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where the two edges meet. Data consisting of vertex coordinates, colours, texture coordinates, etc is associated with a vertex, and each vertex and its associated data are processed independently, in order and in the same way.

Commands are always processed in order in which they are received, although there may be an intermediate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect.

Basic OpenGL Operation

The basic operation of OpenGL is as shown in Figure 3.1(a).

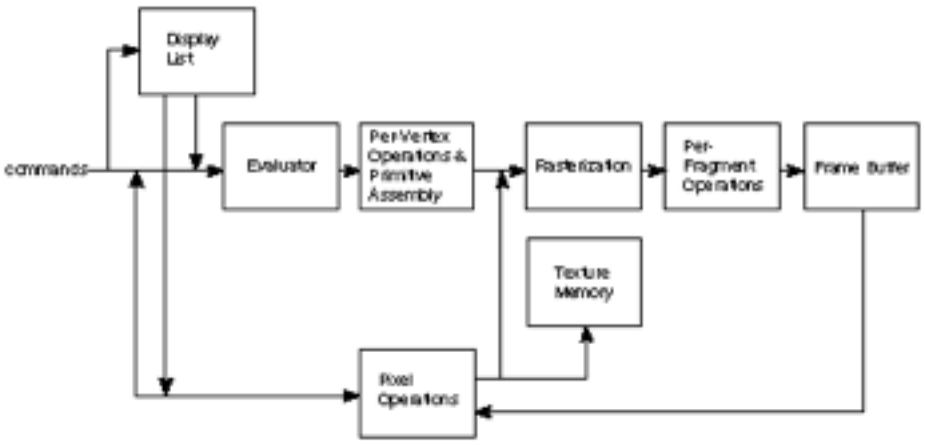


Figure 3.1(a) Basic OpenGL Operation

As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, some of the commands can be accumulated in a display list for processing at a later time.

The evaluator stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. During the next stage, per-vertex operations and primitive assembly, OpenGL processes geometric primitives—points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each fragment produced is fed into the last stage, per-fragment operations, which performs the final operations on the data before it is stored as pixels in the frame buffer.

Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the pixel operations stage. The result of this stage is either stored as texture memory, for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

Conclusion

This activity has presented the definition and fundamentals of OpenGL as an API for creating graphics.

Activity 3.2 – Simple Drawings using OpenGL

Introduction

Every graphics program begins with some initializations to establish the desired display mode and set up a coordinate system for specifying points, lines, etc. Simple graphics drawings use the Cartesian coordinate system with x and y coordinates representing pixels extending to the right and downward/upward directions respectively.

Activity Details

Figure 3.2 (a) shows the different modes in which simple graphics can be drawn. In part (a) the entire screen is used for drawing: the display is initialized by switching it into "graphics mode", and the coordinate system is established as shown. Coordinates x and y are measured in pixels, with x increasing to the right and y increasing downward.

In part (b) a more modern "window-based" system is shown. It can support a number of different rectangular windows on the display screen at one time. Initialization involves creating and "opening" a new window (which can be called the screen window) for graphics. Graphics commands use a coordinate system that is attached to the window: usually x increases to the right and y increases downward. Part (c) shows a variation where the initial coordinate system is "right side up", with y increasing upward.

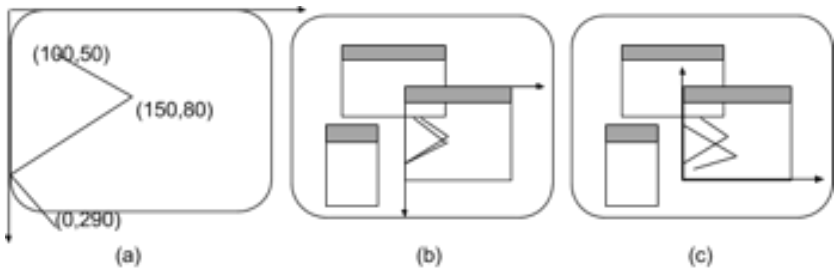


Figure 3.2(a): Display Layouts

Basic Drawing

Each system normally has some elementary drawing tools that help to get started. The most basic has a name like setPixel(x, y, color): it sets the individual pixel at location (x, y) to the color specified by color. It sometimes goes by different names, such as putPixel(), SetPixel(), or drawPoint(). Along with setPixel() there is almost always a tool to draw a straight line, line(x1, y1, x2, y2), that draws a line between (x1, y1) and (x2, y2). In other systems it might be called drawLine() or Line(). The commands

line(100, 50, 150, 80);

line(150, 80, 0, 290);

would draw the pictures shown in each system in Figure 3.2 (a). Other systems have no line() command, but rather use moveto(x, y) and lineto(x, y). They stem from the analogy of a pen plotter, where the pen has some current position. The notion is that **moveto(x, y)** moves the pen invisibly to location (x, y), thereby setting the current position to (x, y); lineto(x, y) draws a line from the current position to (x, y), then updates the current position to this (x, y). Each command moves the pen from its current position to a new position. The new position then becomes the current position. The pictures in Figure 3.2(a) would be drawn using the commands:

moveto(100, 50);

lineto(150, 80);

lineto(0, 290);

In general, a toolkit of functions can be developed from simple elementary tools to build up a library of graphics routines. However, since each graphics display uses different basic commands to "drive it", and every environment has a different collection of tools for producing the graphics primitives, it becomes difficult to port a program from one environment to another, hence this may require major alterations in the overall structure of a library or application, and significant programmer effort.

Event-driven Programming

Most windows-based programs are event-driven; that is the program responds to various events, such as a mouse click, the press of a keyboard key, or the resizing of a screen window. The system automatically manages an event queue, which receives messages that certain events have occurred, and deals with them on a first-come first-served basis. A programmer organizes a program as a collection of callback functions that are executed when events occur. A callback function is created for each type of event that might occur. When the system removes an event from the queue it simply executes the callback function associated with the type of that event.

OpenGL comes with a Utility Toolkit which provides tools to assist with event management. For instance **glutMouseFunc(myMouse)** registers the function **myMouse()** as the function to be executed when a mouse event occurs. The prefix "glut" indicates it is part of the OpenGL Utility Toolkit. The programmer puts code in **myMouse()** to handle all of the possible mouse actions of interest. A skeleton example of a main() for an event-driven function is shown in Figure 3.2(b).

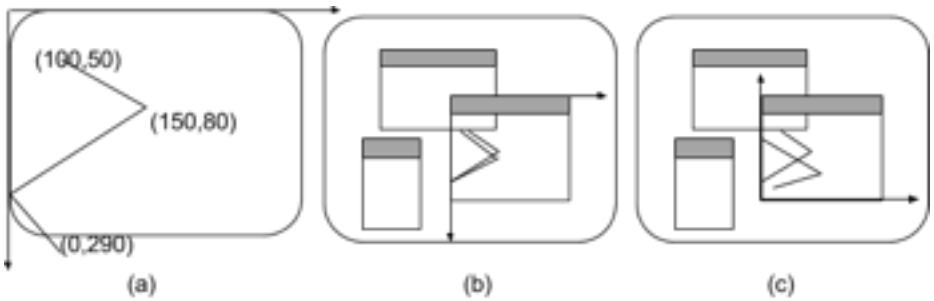


Figure 3.2(b): Code skeleton for main function for an event-driven program using OpenGL

GlutDisplayFunc(myDisplay); Whenever the system determines that a screen window should be redrawn it issues a "redraw" event. This happens when the window is first opened, and when the window is exposed by moving another window off of it. Here the function myDisplay() is registered as the callback function for a redraw event.

GlutReshapeFunc(myReshape); Screen windows can be reshaped by the user, usually by dragging a corner of the window to a new position with the mouse. (Simply moving the window does not produce a reshape event.) Here the function myReshape() is registered with the "reshape" event. As we shall see, myReshape() is automatically passed arguments that report the new width and height of the reshaped window.

GlutMouseFunc(myMouse); When one of the mouse buttons is pressed or released a mouse event is issued. Here myMouse() is registered as the function to be called when a mouse event occurs. myMouse() is automatically passed arguments that describe the mouse location and the nature of the button action.

GlutKeyboardFunc(myKeyboard); This registers the function myKeyboard() with the event of pressing or releasing some key on the keyboard. myKeyboard() is automatically passed arguments that tell which key was pressed. Conveniently, it is also passed data as to the location of the mouse at the time the key was pressed.

GlutMainLoop(). When this is executed the program draws the initial picture and enters an unending loop, in which it simply waits for events to occur. (A program is normally terminated by clicking in the "go away" box that is attached to each window).

Creating Graphics Window

The first task is to open a screen window for drawing. Because OpenGL functions are device independent, they provide no support for window control on specific systems. Figure 3.2(c) shows a skeleton for the entire main() function for a program that will draw graphics in a screen window. The first five function calls use the toolkit to open a window for drawing with OpenGL. The first five functions initialize and display the screen window in which the program will produce graphics. A brief description of what each one does is as follows:

GlutInit(&argc, argv); This function initializes the toolkit. Its arguments are the standard ones for passing command line information.

GlutInitDisplayMode(GLUT_SINGLE | GLUT_RGB); This function specifies how the display should be initialized. The built-in constants GLUT_SINGLE and GLUT_RGB, which are OR'd together, indicate that a single display buffer should be allocated and that colors are specified using desired amounts of red, green, and blue. (Later we will alter these arguments: for example, we will use double buffering for smooth animation.)

GlutInitWindowSize(640,480); This function specifies that the screen window should initially be 640 pixels wide by 480 pixels high. When the program is running the user can resize this window as desired.

GlutInitWindowPosition(100, 150); This function specifies that the window's upper left corner should be positioned on the screen 100 pixels from the left edge and 150 pixels down from the top. When the program is running the user can move this window wherever desired.

GlutCreateWindow("my first attempt"); This function actually opens and displays the screen window, putting the title "my first attempt" in the title bar.

The remaining functions in main() register the callback functions as described earlier, perform any initializations specific to the program at hand, and start the main event loop processing. The programmer implements each of the callback functions as well as myInit().

An example of a complete OpenGL program

The initialization in myInit() sets up the coordinate system, the point size, the background color, and the drawing color. The drawing is encapsulated in the callback function myDisplay(). As this program is non-interactive, no other callback functions are used. glFlush() is called after the dots are drawn to insure that all data is completely processed and sent to the display. This is

important in some systems that operate over a network: data is buffered on the host machine and only sent to the remote display when the buffer becomes full or a glFlush() is executed. The output of the program is as shown in Figure 3.2(e).

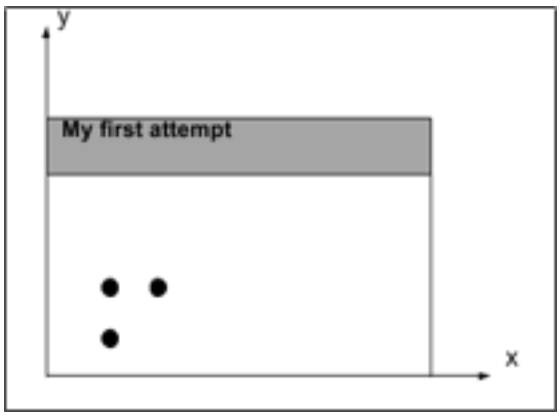


Figure 3.2(e) Drawing three dots using OpenGL

Interaction with Mouse and Keyboard

Data about the mouse is sent to the application by designing the callback function **myMouse()** to take four parameters, so that it has the prototype:

void myMouse(int button, int state, int x, int y);

When a mouse event occurs the system calls the registered function, supplying it with values for these parameters. The value of button will be one of:

GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON, or GLUT_RIGHT_BUTTON,

with the obvious interpretation, and the value of state will be one of: **GLUT_UP** or **GLUT_DOWN.** The values x and y report the position of the mouse at the time of the event.

Note: The x value is the number of pixels from the left of the window as expected, but the y value is the number of pixels down from the top of the window.

Example: Placing dots with the mouse.

Each time a user presses down the left mouse button a dot is drawn in the screen window at the mouse position. If the user presses the right button the program terminates. The version of myMouse() shown next does the this. Because the y-value of the mouse position is the number of pixels from the top of the screen window, a dot is drawn, not at (x, y), but at (x, screenHeight – y), where screenHeight is assumed to be the height of the window in pixels.

```
void myMouse(int button, int state, int x, int y)
```

```
{
if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
drawDot(x, screenHeight -y);
else if(button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
exit(-1);
}
```

Interaction with Keyboard

Pressing a key on the keyboard queues a keyboard event. The callback function myKeyboard() is registered with this type of event through **glutKeyboardFunc(myKeyboard)**. It must have prototype:

void myKeyboard(unsigned int key, int x, int y);

The value of key is the ASCII value of the key pressed. The values x and y report the position of the mouse at the time that the event occurred. (As before y measures the number of pixels down from the top of the window).

The programmer can capitalize on the many keys on the keyboard to offer the user a large number of choices to invoke at any point in a program. Most implementations of myKeyboard() consist of a large switch statement, with a case for each key of interest. An example is shown below, where pressing 'p' draws a dot at the mouse position; pressing the left arrow key adds a point to some (global) list, but does no drawing; pressing 'E' exits from the program. Note that if the user holds down the 'p' key and moves the mouse around a rapid sequence of points is generated to make a "freehand" drawing.

```
void myKeyboard(unsigned char theKey, int mouseX, int mouseY)
{
GLint x = mouseX;
GLint y = screenHeight - mouseY; // flip the y value as always
switch(theKey)
{
case 'p':
drawDot(x, y); // draw a dot at the mouse position
break;
case GLUT_KEY_LEFT: List[++last].x = x; // add a point
List[ last].y = y;
break;
case 'E':
exit(-1); //terminate the program
default:
break; // do nothing
}
}
```

Conclusion

OpenGL is an API which provides useful routines for drawing simple graphics. This activity has presented the basics of drawing simple graphics using OpenGL. Most graphics applications are written for a windows-based environment. The program opens a window on the screen that can be moved and resized by the user, and it responds to mouse clicks and keyboard strokes.

Activity 3.3 – Viewports, Mapping and Clipping

Introduction

In a given problem, it may be much more natural to think in terms of x varying from, say, -1 to 1, and y varying from –100.0 to 20.0 and not only the positive values of x and y. This activity presents methods that let a programmer describe objects in whatever coordinate system best fits the problem at hand, and to have the picture automatically scaled and shifted so that it fits on the screen window.

Activity Details

Viewports

In the previous activity, drawings used the basic coordinate system of the screen window: coordinates that are essentially in pixels, extending from 0 to some value screenWidth – 1 in x, and from 0 to some value screenHeight –1 in y. This means that only positive values of x and y can be used, and the values must extend over a large range (several hundred pixels) if to get a drawing of some reasonable size. In a given problem, it may be much more natural to think in terms of x varying from, say, -1 to 1, and y varying from –100.0 to 20.0. A picture can be automatically scaled and shifted so that it fits in the screen window.

The space in which objects are described is called world coordinates. It is the usual Cartesian xy-coordinate system used in mathematics, based on whatever units are convenient. A rectangular world window is defined in these world coordinates. The world window specifies which part of the "world" should be drawn. The understanding is that whatever lies inside the window should be drawn; whatever lies outside should be clipped away and not drawn. In addition, we define a rectangular viewport in the screen window on the screen. A mapping (consisting of scalings and shiftings) between the world window and the viewport is established so that when all the objects in the world are drawn, the parts that lie inside the world window are automatically mapped to the inside of the viewport. So a programmer thinks in terms of "looking through a window" at the objects being drawn, and placing a "snapshot" of whatever is seen in that window into the viewport on the display. This window/viewport approach makes it much easier to do natural things like "zooming in" on a detail in the scene, or "panning around" a scene.

World Windows and Viewport

An example is used to illustrate the use of windows and viewports. Sinc(x), a famous function in signal processing, is expressed as follows:

$$\sin c(x) = \frac{\sin(\pi x)}{\pi x}$$

As x varies from -∞ to ∞the value of sinc(x) varies over a range of -1 to 1. For instance, a plot of sinc(x) that is centred at (0,0) and showing sinc(x) for closely spaced x values between, say -4.0 and 4.0 can be drawn using OpenGL to generate a plot as shown in Figure 4.3(a).

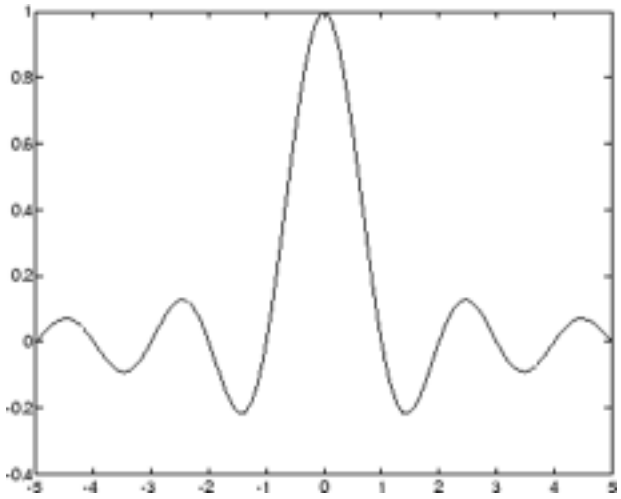


Figure 4.3(a). Sinc(x) Plot

The OpenGL display function code is as follows:

```
void myDisplay(void)
{
glBegin(GL_LINE_STRIP);
for(GLfloat x = -4.0; x < 4.0; x += 0.1)
{
GLfloat y = sin(3.14159 * x)/(3.14159 * x);
glVertex2f(x, y);
}
glEnd();
glFlush();

}
```

The code in these examples operates in a natural coordinate system for the problem: x is made to vary in small increments from –4.0 to 4.0. The key issue here is how the various (x, y) values become scaled and shifted so that the picture appears properly in the screen window.

Window to Viewport Mapping

Window to viewport mapping is used to map world coordinates to screen or device coordinates. A window is an area that defines what is to be displayed, while a viewport is the area that defines where it is to be displayed. Figure 4.3(b) shows a window in world coordinates and the viewport in screen or display coordinates. Whatever area is selected in the window is sent to the viewport. The size of the viewport might be smaller or larger than the window.

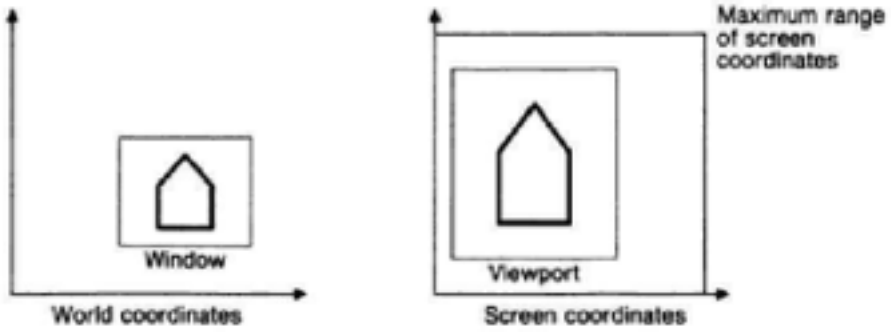


Figure 4.3(b): The window in world coordinates and the viewport in screen coordinates

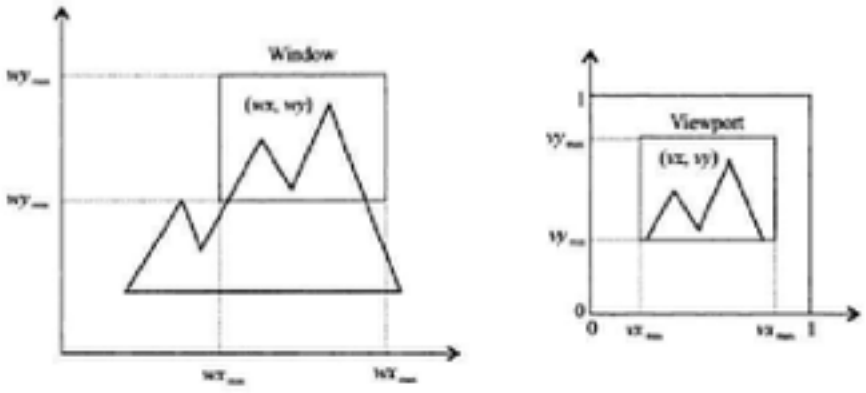


Figure 4.3(c): Window to viewport transformation example

The window-to-viewport transformation maintains the relative position of a point in window as well as in the viewport. A point at position (xw,yw) in the window is mapped into position (xv,yv) in the associated viewport. An example is shown in figure 4.3(c).

Clipping

Any part of a picture that lies beyond the confines of the screen cannot be

displayed, hence methods to handle pictures that are larger than available display screen size are used. Suppose the size of the picture to be shown is bigger than the size of the screen, then only a portion of the picture can be displayed. The context is similar to that of viewing a scene outside the window. While the scene outside is quite large, a window will allow to see only that portion of the scene as can be visible from the window – the latter is limited by the size of the window. If we presume that the screen, which allows us to see the pictures as a window, then any picture whose parts lie outside the limits of the window cannot be shown and for algorithmic purposes, they have to be "clipped".

Conclusion

This activity has presented how objects in one coordinate system can be mapped into to another, i.e. world to screen/device coordinates. This allows the picture to be scaled and shifted so that it can fit on the device window.

Unit Summary

This unit has covered the basic aspects in picture drawing using OpenGL as an API. Simple OpenGL programs are also presented for simple picture drawings such as lines and rectangles. Picture transformation from window to viewport is also presented, where the mapping of a picture from one coordinate system to a required coordinate system is performed. The unit also presented how pictures can be clipped in order to fit on display devices.

Unit Assessment

Instructions

Assessment

1. Define OpenGL.
2. Describe the way in which drawing simple graphics using OpenGL can be achieved.
3. Briefly explain the Rasterization process.
4. Write a program in OpenGL to draw the following:
   a. Draw a rubber rectangle,
   b. Dotted lines by mouse.
5. Define window and viewpoint.

6. Define clipping.
7. Explain why clipping is important in drawing graphics.
8. Describe how window to viewport mapping is performed.
9. Write an OpenGL program to find the area of a circle.
10. Write an OpenGL program which takes in students grades and calculate average.

Grading Scheme

Marks are indicated next to each question.

Answers

These are solutions to the above questions

1. OpenGL is an application programming interface (API) for creating 2D and 3D graphic images
2. Drawing can be achieved through the following stages i.e The evaluator stage, per-vertex operations and primitive assembly stage, Rasterization stage, per-fragment operations stage and frame buffer.
3. Rasterization is a process which produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon.
4. Window is a world-coordinate area selected for display and viewpoint is an area on a display device to which a window is mapped.
5. Clipping is the limitation of see only the part of the image which is within the window size
6. Clipping is important in handling huge images
7. The window to viewport mapping involves three step transformation composition which are window translated to origin, window scaled to size of viewport and window translated by some matrix to the final position.

Unit 4. 2D and 3D Graphics

Unit Introduction

This topic will introduce you to the basic principles on viewing objects in two Dimensions and three Dimensions. The topic will also cover the various physical devices which are available for data input on graphics workstations with their logical classification.

Unit Objectives

Upon completion of this unit you should be able to:

- Demonstrate how to view objects in two dimensions
- Describe the basic input devices in graphics
- Demonstrate how to view and transform objects in three dimensions

Key Terms

2D: Two-dimensional

3D: Three-dimensional

World coordinate: Dimensional coordinates that define the location of objects in the real world.

Screen coordinates: Physical coordinates of the pixels on the computer screen, based on current screen resolution.

Transformation: changing some graphics into something else by applying rules

Translation: shifting of a point to some other place, whose distance with regard to the present point is known

Rotation: rotation of a point about an axis

Learning Activities

Activity 4.1: Two Dimensions Viewing

Introduction

This section will introduce you to the formal mechanism for displaying views of a picture on an output device. Any convenient Cartesian coordinate system, referred to as the world-coordinate reference frame, can be used to define the picture. For a two-dimensional picture, a view is selected by specifying a subarea of the total picture area. Transformations from world to device coordinates involve translation, rotation, and scaling operations, as well as procedures for deleting those parts of the picture that are outside the limits of a selected display area.

Activity Details

A world-coordinate area selected for display is called a window. An area on a display device to which a window is mapped is called a viewport. The window defines what is to be viewed; the viewport defines where it is to be displayed. Often, windows and viewports are rectangles in standard position, with the rectangle edges parallel to the coordinate axes. Other window or viewport geometries, such as general polygon shapes and circles, are used in some applications, but these shapes take longer to process. In general, the mapping of a part of a world-coordinate scene to device coordinates is referred to as a viewing transformation. Sometimes the two-dimensional viewing transformation is simply referred to as the window-to-viewport transformation or the windowing transformation.

But, in general, viewing involves more than just the transformation from the window to the viewport. Figure 4.1.1 illustrates the mapping of a picture section that falls within a rectangular window onto a designated &angular viewport. In computer graphics terminology, the term window originally referred to an area of a picture that is selected for viewing, as defined at the beginning of this section. Unfortunately, the same tern is now used in window-manager systems to refer to any rectangular screen area that can be moved about, resized, and made active or inactive. In this chapter, we will only use the term window to refer to an area of a world-coordinate scene that has been selected for display.

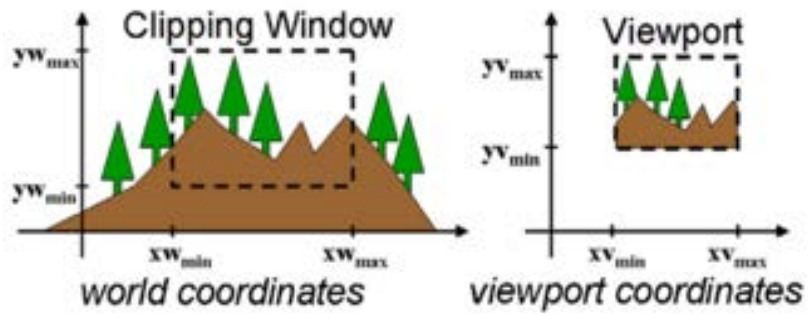
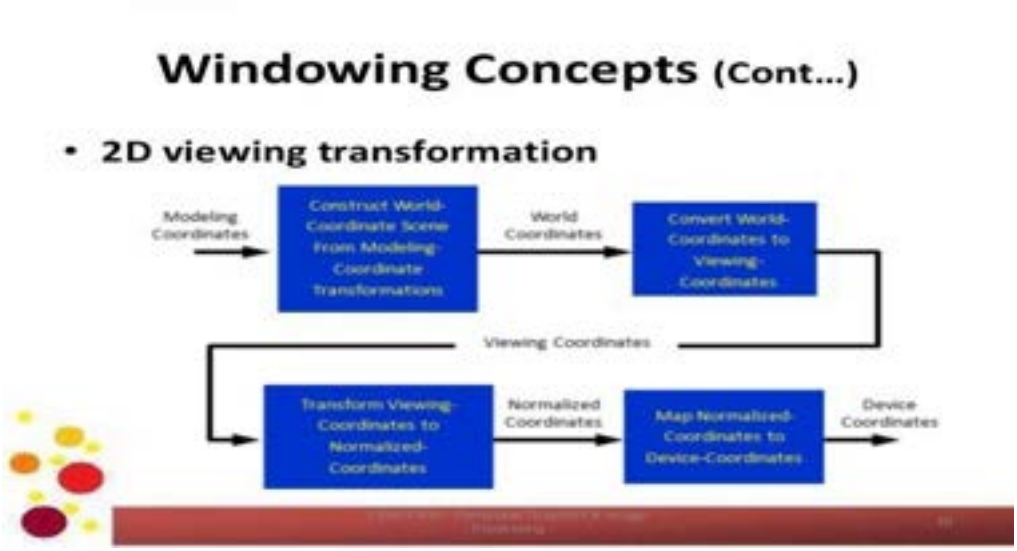


Figure 4.1.1: A viewing transformation using standard rectangles for the window and viewport.

Some graphics packages that provide window and viewport operations allow only standard rectangles, but a more general approach is to allow the rectangular window to h a w any orientation. In this case, we carry out the viewing transformation in several steps, as indicated in Figure 4.1.2 below



4.1.2: The two-dimensional viewing-transformation pipeline.

Conclusion

In this section, we have seen how we can map a two-dimensional world coordinate scene to a display device. The viewing-transformation pipeline includes constructing the scene using modeling transformations transferring world-coordinates to viewing coordinates, mapping the viewing coordinate description objects to normalized coordinates, and finally mapping to device coordinates. Normalized coordinates are specified in the range from 0 to 1, and are used to make viewing packages independent of particular output device.

Activity 4.2 Display Tools

Introduction

This section will introduce you to the various physical devices which are available for data input on graphics workstations. These devices will be grouped based on their logical classification.

Activity Details

Keyboard is an alphanumeric keyboard on a graphics system is used primarily as a device for entering text strings. The keyboard is an efficient device for inputting such non graphic data as picture labels associated with a graphics display. Keyboards can also be provided with features to facilitate entry of screen coordinates, menu selections, or graphics functions.

A mouse is small hand-held box used to position the screen cursor. Wheels or rollers on the bottom of the mouse can be used to record the amount and direction of movement. Another method for detecting mouse motion is with an optical sensor. For these systems, the mouse is moved over a special mouse pad that has a grid of horizontal and vertical lines. The optical sensor detects movement across the lines in the grid.

Trackball is a ball that can be rotated with the fingers or palm of the hand, to produce screen-cursor movement. Potentiometers, attached to the ball, measure the amount and direction of rotation. Trackballs are often mounted on keyboards or other devices such as the Z mouse.

While a trackball is a two-dimensional positioning device, a space ball provides six degrees of freedom. Unlike the trackball, a space ball does not actually move. Strain gauges measure the amount of pressure applied to the space ball to provide input for spatial positioning and orientation as the ball is pushed or pulled in various directions. **Space balls** are used for three-dimensional positioning and selection operation

A **joystick** consists of a small, vertical lever (called the stick) mounted on a base that is used to steer the screen cursor around. Most joysticks select screen positions with actual stick movement; others respond to pressure on the stick. Some joysticks are mounted on a keyboard; others function as stand-alone units.

Digitizers are a common device for drawing, painting, or interactively selecting coordinate positions on an object is a digitizer. These devices can be used to input coordinate values in either a two-dimensional or a three-dimensional space. Typically, a digitizer is used to scan over a drawing or object and to input a set of discrete coordinate positions, which can be joined with straight line segments to approximate the curve or surface shapes.

Drawings, graphs, color and black-and-white photos, or text can be stored for computer processing with an image scanner by passing an optical scanning mechanism over the information to be stored. The gradations of gray scale or color are then recorded and stored in an array. Once we have the internal representation of a picture, we can apply transformations to rotate, scale, or crop the picture to a particular screen area.

Touch Panels allow displayed objects or screen positions to be selected with the touch of a finger. A typical application of touch panels is for the selection of processing options that are represented with graphical icons.

Light Pens are pencil-shaped devices are used to select screen positions by detecting the light coming from points on the CRT screen. They are sensitive to the short burst of light emitted from the phosphor coating at the instant the electron beam strikes a particular point.

Voice Systems are speech recognizers are used in some graphics workstations as input devices to accept voice commands. The voice-system input can be used to initiate graphics operations or to enter data. These systems operate by matching an input against a predefined dictionary of words and phrase.

Logical Classification of the input devices:

Locator Devices are used to provide a coordinate position at the desired location. Common devices are thumbwheel, trackball, joystick, mouse and touch panels.

Stoke devices used to provide a series of coordinated positions. It's essentially multiple calls to a locator's device. Common devices are tablet and light pen.

String devices are used to provide text. The common string device is the keyboard. Other devices are stroke devices with character recognizers.

Valuators devices are used to provide scalar values of rotation angles, scale factors, application parameters. Common valuators devices are keyboard (numeric input)

Choice device are used to select menu options. Common choice devices are touch panels, light pens

Pick devices are used to select picture components. A common pick device is the light pen.

Conclusion

In this chapter, we have seen for graphical input, we have a range of devices to choose from and how these devices can be categories logically.

Activity 4.3: Three Dimensional Translation

Introduction

Methods for geometric transformations and object modeling in three dimensions are extended from two-dimensional methods by including considerations for the z coordinate. We now translate an object by specifying a three dimensional translation vector, which determines how much the object is to be moved in each of the three coordinate directions.

Activity Details

Translation

In a three-dimensional homogeneous coordinate representation, a point is translated from position P = (x, y, z) to position P' = (x ' , y', z'). Parameters tx, ty, and tz, specifying translation distances for the coordinate directions x , y, and z, are assigned any real values. The matrix representation is equivalent to the three equations below

x'=:r+ tx, y'=y+ ty, z ' = z + tz;

Rotation

To generate a rotation transformation for an object, we must designate an axis of rotation (about which the object is to be rotated) and the amount of angular rotation. Unlike two-dimensional applications, where all transformations are carried out in the xy plane, a three-dimensional rotation can be specified around any line in space. The easiest rotation axes to handle are those that are parallel to the coordinate axes. Also, we can use combinations of coordinate axis rotations (along with appropriate translations) to specify any general rotation.

Scaling

The matrix expression tor the scaling transformation of a position P = (x,y,z) relative to the coordinate origin can be written as shown below where scaling parameters sx , sy, and sz, are assigned any positive values.

$$P' = S.P$$

Explicit expressions for the coordinate transformations for scaling relative to the origin are

$x' = x \cdot sx,\ y' = y \cdot sy,\ z' = Z \cdot Sz,$

Scaling an object with transformation changes the size of the object and repositions the object relative to the coordinate origin. Also, if the transformation parameters are not all equal, relative dimensions in the object are changed.

Conclusion

Representations for translation and scaling are straightforward extensions of two-dimensional transformation representations. For rotations, however, we need more general representations, since objects can be rotated about any specified axis in space. Any three-dimensional rotation can be represented as a combination of basic rotations around the x, y, and z axes.

Activity 4.4: Three Dimensions Viewing

Introduction

In two-dimensional graphics applications, viewing operations transfer positions from the world-coordinate plane to pixel positions in the plane of the output device. For three-dimensional graylucs applications, the situation is a bit more involved, since we now have more choices as to how views are to be generated. First of all, we can view an object from any spatial position: from the front, from above, or from the back.

Activity Details

The steps for computer generation of a view of a three-dimensional scene are somewhat analogous to the processes involved in taking a photograph. To take a snapshot, we first need to position the camera at a particular point in space. Then we need to decide on the camera orientation (c.f. Figure 4.4.1).

Figure 4.4.1: Photographing a scene involves selection of a camera position and orientation

Which way do we point the camera and how should we rotate it around the line of sight to set the u p direction for the picture? Finally, when we snap the shutter, the scene is cropped to the size of the "window" (aperture) of the camera, and light from the visible surfaces is projected onto the camera film. We need to keep in mind, however, that the camera analogy can be carried only so far, since we have more flexibility and many more options for generating views of a scene with a graphics package than we do with a camera.

Figure 4.4.2 shows the general processing steps for modeling and converting a world-coordinate description of a scene to device coordinates. Once the scene has been modeled, world-coordinate positions are converted to viewing coordinates. The viewing-coordinate system is used in graphics packages as a reference for specifying the observer viewing position and the position of the projection plane, which we can think of in analogy with the camera film plane. Next, projection operations are performed to convert the viewing-coordinate description of the scene to coordinate positions on the projection plane, which will then be mapped to the output device. Objects outside the specified viewing limits are clipped h m further consideration, and the remaining objects are processed through visible-surface identification and surface-rendering procedures to produce the display within the device viewport.

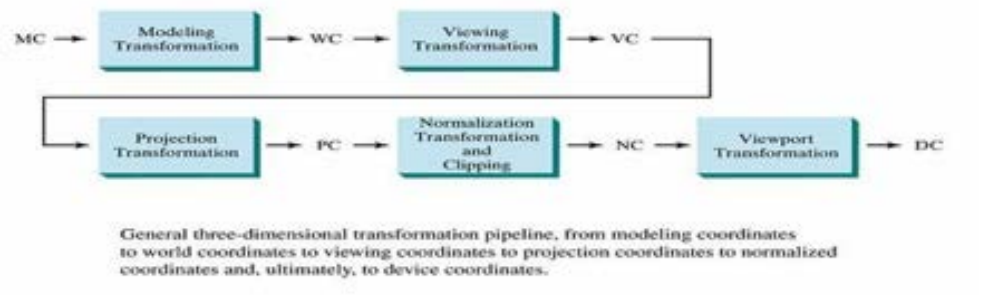


Figure 4.4.2: General Three dimension transformation pipeline, from modeling coordinates to final devices coordinates

Conclusion

Viewing procedures for three-dimensional scenes follow the general approach used in two-dimensional viewing. That is, we first create a world-coordinate scene from the definitions of objects in modeling coordinates. Then we set up a viewing coordinate reference frame and transfer object descriptions from world coordinates to viewing coordinates. Finally, viewing coordinate descriptions are transformed to device coordinates.

Unlike two-dimensional viewing, however, three-dimensional viewing requires projection routines to transform object descriptions to a viewing plane before the transformation to device coordinates. Also three-dimensional viewing operations involve more spatial parameters. We can use the camera analogy to describe three-dimensional viewing parameters, which include camera position and orientation.

Unit Summary

This unit has presented the basic principles of viewing objects in two and three dimensions. Various physical devices for data input on graphics workstations have been presented, together with their logical classification. Methods for geometric transformations and object modeling in three dimensions have also been presented whereby transformations are extended from two-dimensional methods by including considerations for the z coordinate.

Unit Assessment

Instructions

Answer the following questions

1. List all the devices based on their logical classification. (3 marks)
2. Identify the locators devices which can also be used as the stoke devices. (2 marks)
3. Give more details on the Voice Systems. (3 marks)
4. Mention any three methods which can be used to transforms objects in three dimensions. (3 marks)
5. Give the location of the point (X,Y,Z) which is translated by amount Dx, Dy and Dz. (2 marks)
6. In OpenGL translation, rotation, and scaling are performed using which commands? (2 marks)
7. Explain briefly how an object can be transformed from the World coordinated to the Device Coordinated in a three dimension Viewing. (3 marks)

Grading Scheme

Marks are indicated next to each question.

Answers

These are solutions to the above questions

Solutions

1. Locator Devices, Stoke devices, String devices, Valuators devices, Choice device and Pick devices
2. Touch panels.
3. Voice Systems are speech recognizers are used in some graphics workstations as input devices to accept voice commands
4. Translation, rotation and Scaling
5. (Dx + X,Dy + Y,Dz + Z)
6. glTranslate{fd}(X,Y,Z) - glTranslatef(1.0, 2.5, 3.0)

   glRotate{df}(Angle, X, Y, Z) - glRotatef(60.0, 0.0, 0.0, 1.0)

   glScale{df}(X, Y, Z) - glScalef(1.0, 1.5, 2.0)
7. Conversion of objection descriptions from world to viewing coordinates is equivalent to a transformation that superimposes the viewing reference frame onto the world frame using the basic geometric translate-rotate operations:

   a. Translate the view reference point to the origin of the world-coordinate system.

   b. Apply rotations to align the xv, yv, and zv axes (viewing coordinate system) with the world xw, yw, zw axes, respectively.

Course Summary

This course has introduced various topics for the basic understanding and application of computer graphics. The basic concepts of digital images and how colour can be presented in computers has been addressed. Fundamental operations performed in computer graphics, such as translation, transformation and rotation, have been presented. These operations enable images appear different from what they actually look like. The 2D pipeline has also been presented, whereby 2D world coordinate system images are mapped to device coordinates to allow viewing at different device settings. The API for use in producing graphics, the 4GL, has been presented, where sample programs that produce various pictures are given.

The African Virtual University Headquarters

Cape Office Park

Ring Road Kilimani

PO Box 25405-00603

Nairobi, Kenya

Tel: +254 20 25283333

contact@avu.org

oer@avu.org

The African Virtual University Regional Office in Dakar

Université Virtuelle Africaine

Bureau Régional de l'Afrique de l'Ouest

Sicap Liberté VI Extension

Villa No.8 VDN

B.P. 50609 Dakar, Sénégal

Tel: +221 338670324

bureauregional@avu.org