

MODULE 5: INTRODUCTION TO HTML AND WEBSITE DEVELOPMENT

PART I: THE WORLD WIDE WEB AND HYPERTEXT

4.3 THE WORLD WIDE WEB (from Computer Science: An Overview)

Origin and Development

The World Wide Web had its origins in the work of **Tim Berners-Lee** who realized the potential of combining internet technology with the concept of linked documents called **hypertext**. His first software for implementing the Web was released in December 1990. While this early prototype did not yet support multimedia data, it included the key components of what we now recognize as the World Wide Web:

1. **Hypertext document format** for embedding hyperlinks to other documents
2. **Protocol for transferring hypertext** across the network
3. **Server process** that supplied hypertext pages upon request

With this, the Web quickly grew to support images, audio, and video. By the mid-1990s, it had become the dominant application powering the growth of the Internet.

Web Implementation

Two Categories of Software Packages:

Type	Description	Examples
Browser	Resides on user's computer; obtains materials requested by user; presents materials in organized manner	Firefox, Safari, Internet Explorer, Chrome
Webserver	Resides on computer containing hypertext documents; provides access to documents under its control as requested by clients	Apache, IIS, Nginx

HTTP (Hypertext Transfer Protocol):

- Protocol used to transfer hypertext documents between browsers and web servers

URL (Uniform Resource Locator)

Definition: Unique address for each document on the Web. Contains information needed by browser to contact proper server and request the desired document.

Components of a Typical URL:

Segment	Purpose
Protocol	Protocol to communicate with server controlling access to document
Machine Address	Mnemonic address of machine containing the server
Directory Path	Path needed for server to find directory containing document
Document Name	Name of the document itself

Shortened URLs:

- May not explicitly contain all segments
- No directory path if server doesn't need it
- URL consisting of only protocol and mnemonic address → webserver returns predetermined document (**home page**)

Example: `http://www.google.com` → leads to Google home page

Browser Assumption: Many browsers assume HTTP protocol should be used if no protocol is identified. They correctly retrieve Google home page when given URL consisting merely of `www.google.com`.

HTML (Hypertext Markup Language)

Definition: Traditional hypertext document is similar to a text file because its text is encoded character by character using a system such as ASCII or Unicode. The distinction is that a hypertext document also contains special symbols called **tags** that describe how the document should appear on a display screen, what multimedia resources should accompany the document, and which items within the document are linked to other documents.

Purpose of HTML Tags:

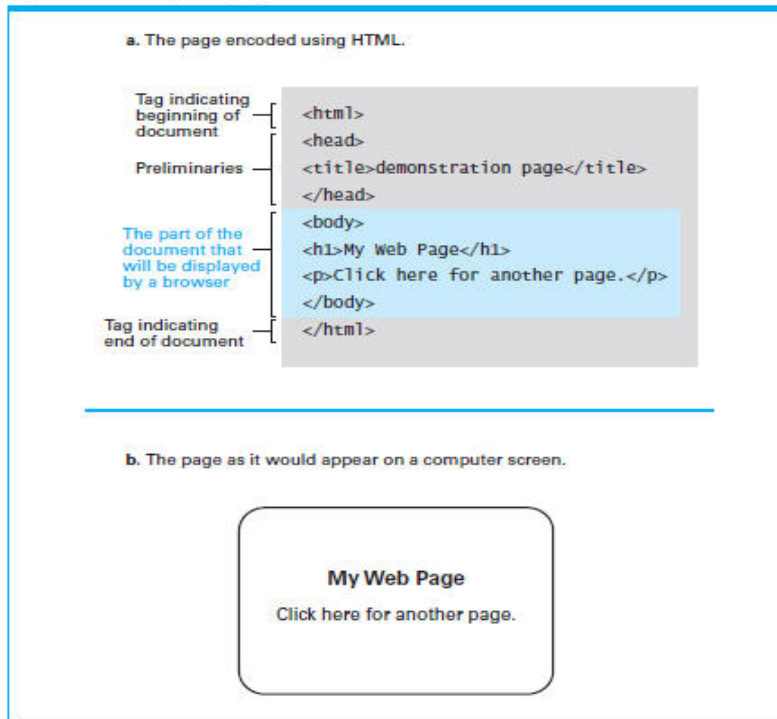
- Describe how document should appear on display screen
- Indicate what multimedia resources (images) should accompany document
- Identify which items within document are linked to other documents

Adding typesetting directions to plain typed text (using red pen) so that a typesetter knows how the material should appear in its final form. In hypertext, the red markings are replaced by HTML tags, and a browser ultimately plays the role of the typesetter.

Simple Webpage Example:

Simple Webpage is shown in FIGURE 4.9

Figure 4.9 A simple webpage



Document Structure:

Section	Tags	Purpose
Head	<code><head> </head></code>	Preliminary information about document (title)
Body	<code><body> </body></code>	Material to be presented on computer screen

Common Tags:

Tag	Meaning	Description
<code><h1> </h1></code>	Level-one heading	Display text prominently
<code><p> </p></code>	Paragraph	Block of text
<code><title> </title></code>	Title	Documentation only; not displayed on screen

Adding Hyperlinks - Anchor Tags:

Purpose: Link text to another document

HTML Code:

text

```
<a href="http://crafty.com/demo.html">here</a>
```

Parameter: href = "http://crafty.com/demo.html"

- href = hypertext reference
- URL following equal sign = document to retrieve when link clicked

Result: When clicked on highlighted term, browser retrieves and displays the associated webpage. It is by means of anchor tags that webpages are linked to each other.

Adding Images:

HTML Code:

text

```

```

Parameter: src = "source"

- Information following equal sign indicates source of image to be displayed

Placement:

- Immediately after <body> tag → image at top of webpage
- Just before </body> tag → image at bottom of webpage

Process:

1. Browser finds image tag
2. Sends message to HTTP server requesting the image
3. Displays image appropriately

XML (eXtensible Markup Language)

Concept:

- HTML = notational system for encoding text document + appearance as simple text file
- Similarly, we can encode nontextual material as text files
- Example: Sheet music can be encoded using custom tags

Music Encoding Example:

text

```
<staff clef="treble">
  <key>C minor</key>
  <time>2/4</time>
  <measure>
    <rest>egh</rest>
    <notes>egh G, egh G</notes>
  </measure>
  <measure>
    <notes>hlf E</notes>
  </measure>
</staff>
```

Style Similarities with HTML:

- Tags delineated by < and >
- Beginning and end of structures indicated by tags of same name
- Ending tag designated by slash (</measure>)
- Special attributes within tags: clef="treble"

XML Definition:

XML is a standardized style (similar to the music example) for designing notational systems for representing data as text files. Actually, XML is a simplified derivative of an older set of standards called SGML (Standard Generalized Markup Language).

Applications of XML:

- Mathematics
- Multimedia presentations
- Music
- HTML is the markup language based on XML standard developed for representing webpages

Note: Original HTML was developed before XML standard was solidified, therefore some features of HTML do not strictly conform to XML. **XHTML** is the version of HTML that rigorously adheres to XML.

Significance of XML:

XML provides a good example of how standards are designed to have wide-ranging applications. Rather than designing individual, unrelated markup languages for encoding various types of documents, the approach represented by XML is to develop a standard for markup languages in general. Markup languages developed in this manner possess a uniformity that allows them to be combined to obtain markup languages for complex applications.

Semantic vs. Appearance Markup:

Aspect	HTML (Appearance-oriented)	XML (Semantic-oriented)
Focus	How content should look	What content means
Tags	Describe presentation	Describe meaning
Example	<h1> for big text	<ingredient> for recipe ingredient
Search	Find words	Find concepts

World Wide Semantic Web:

By using an Internet-wide standard for marking documents according to semantics rather than appearance, a World Wide Semantic Web (rather than the World Wide Syntactic Web we have today) would be created.

Client-Side and Server-Side Activities

Simple Webpage Retrieval Process:

1. Browser (client) uses URL to contact webserver
2. Requests copy of page
3. Server responds by sending text document (HTML source)
4. Browser interprets HTML tags
5. Displays page on screen
6. User clicks link → browser uses URL in anchor tag to contact appropriate server
7. Obtain and display another webpage

Type	Performed By	Description
Client-Side Activities	Client (browser)	Executed on user's computer
Server-Side Activities	Server (webserver)	Executed on server computer
Examples:		
Application	Client-Side Activity	Server-Side Activity
Travel Agent	Customer specifies destinations/dates	Construct customized webpage; send to customer
Search Engine	User specifies topic of interest	Construct customized webpage identifying documents of possible interest
Web Mail	User creates messages; browser displays mail	Build webpages with information from mail server; forward messages to mail server

Client-Side Technologies:

Technology	Description	Developer
JavaScript	Programs included within HTML source document; browser extracts and follows as needed	Netscape Communications, Inc.
Applets	Program units transferred to browser as requested within HTML source document	Sun Microsystems (Java)
Flash	Extensive multimedia client-side presentations	Macromedia

Server-Side Technologies:

Technology	Description	Developer
CGI (Common Gateway Interface)	Clients request execution of programs stored at server	Standards-based

Technology	Description	Developer
Servlets	Clients cause program units to be executed at server side	Sun Microsystems
JSP (JavaServer Pages)	Webpage templates completed using information received from client	Sun Microsystems
ASP (Active Server Pages)	Templates for constructing customized webpages	Microsoft
PHP	Open source system for server-side functionality	Open source community

Security and Ethical Problems:

- **Client side:** Client blindly executes any program sent by webserver → opens itself to malicious activities
- **Server side:** Clients can cause programs to be executed at server → security breaches and potential damage at server can result

PART II: INTRODUCTION TO HTML AND WEBSITE DEVELOPMENT

(From Fundamentals of Information Technology - Chapter 12)

12.1 OVERVIEW

Historical Context

Before the Web (Early Internet):

- Internet only supported plain text
- Required users to learn precise syntax of several computer commands
- Example: To send email to jane@example.com required typing:

```
mail -s "mail subject" jane@example.com <<< "Mail body"
```

- Most users found this too challenging
- Only users on early Internet were computer science researchers at universities

The Birth of the Web - 1989:

- **Sir Tim Berners Lee** proposed and demonstrated an information management system based on **hyperlinks**

- Users could click on links to navigate to other pages
- By 1993, Berners Lee had specified the first version of HTML

The Mosaic Browser - 1992-1993:

- **Marc Andreessen** (student) and **Eric Bina** (graduate) at University of Illinois at Urbana-Champaign
- Working at National Center for Supercomputing Applications (NCSA)
- Built the Mosaic web browser
- Added the tag to HTML to handle images
- Released the web browser to the Internet
- Browser became immensely popular - acquired status of a **killer application** that popularized the Internet

Today:

- Over 5 billion people (over 60% of world's population) use the Internet
- HTML and related technologies have made it easy, inexpensive, and meaningful for anyone anywhere in the world to access the web

12.2 WHAT IS HTML?

HTML (Hypertext Markup Language):

- The standard markup language used for creating webpages and web applications
- A **markup language** is a computer language used to apply layout and formatting conventions to a text document
- Makes text on webpages more interactive and dynamic
- Allows specification of placement of images, tables, links, etc. using simple text descriptions
- Provides the structure and content of a webpage
- Specifies how text, images, hyperlinks, tables, and other media should be displayed on a browser

Hypertext:

- Means "text within text"
- In context of World Wide Web, refers to text with a link to another page within it
- Internet's way to link two or more webpages (HTML documents) with each other

Viewing HTML Source:

- Right-click on any webpage → "View Page Source"
- Displays the text specification of that webpage
- All common browsers (Google Chrome, Apple Safari, Microsoft Internet Explorer, Mozilla Firefox) understand HTML

Key Characteristics:

- HTML has been an Internet standard since 1993
- Uses **tags** to define elements
- Tags can have **attributes** to further describe elements
- Simplicity makes it easy even for humans to understand the content
- Foundational technology that all modern professionals will find useful

12.3 STRUCTURE OF AN HTML PAGE

Basic Principle: HTML encloses text within tags to tell the browser how to render the enclosed text.

Examples:

- `<b>this is bold text</b>` → **this is bold text**
- `<i>this is italicized text</i>` → *this is italicized text*

File Extension: HTML pages are saved with .html extension

Basic HTML Page Structure:

text

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>Page Title</title>
```

```
  </head>
```

```
  <body>
```

```
    <h1>This is a Heading</h1>
```

```
    <p>This is a paragraph.</p>
```

```
  </body>
```

```
</html>
```

Component	Description
DOCTYPE Declaration	First line of HTML page; tells browser what kind of document to expect, particularly which version of HTML
HTML tag (<html> </html>)	Defines start and end of HTML document; everything nested within these tags
Head tag (<head> </head>)	Contains information about HTML document mostly not displayed to user; includes title, meta information, stylesheets, JavaScript
Title tag (<title> </title>)	Only content within head displayed to user; appears as title of browser tab
Body tag (<body> </body>)	Contains all content displayed to user; includes text, images, links, and other HTML elements

12.4 ELEMENTS IN HTML

Essential HTML Tags

1. Paragraph Tag (<p>):

- Defines a block of text as a paragraph
- Text within <p> tag displayed as separate paragraph with line break before and after
- Can include text, images, links, or other HTML elements

Example:

html

```
<p>This is a paragraph.</p>
```

```
<p style="color:red;">This is a red paragraph.</p>
```

2. Heading Tags (<h1> to <h6>):

- Create structure within pages
- <h1> represents main heading on a page (typically used for page title)
- <h2> to <h6> used for subheadings

- Best practice: Do not skip heading levels (subsections of <h1> should use <h2>, not <h3>)

Example:

html

```
<h1>Main Title</h1>
```

```
<h2>Section Title</h2>
```

```
<h3>Subsection Title</h3>
```

3. Text Formatting Tags:

Tag	Purpose	Example
<code> </code>	Bold text	<code>bold</code> → bold
<code><i> </i></code>	Italic text	<code><i>italic</i></code> → <i>italic</i>
<code><u> </u></code>	Underline text	<code><u>underlined</u></code> → <u>underlined</u>
<code></code> or <code><strike></code>	Strikethrough text	<code>deleted</code> → ~~deleted~~
<code><sub> </sub></code>	Subscript	H <code><sub>2</sub></code> O → H ₂ O
<code><sup> </sup></code>	Superscript	E=mc <code><sup>2</sup></code> → E=mc ²

Adding Hyperlinks

Anchor Tag (<a>):

- Used to add hyperlinks to an HTML page
- Has required attribute href to specify target page
- Target page specified using URL of the resource to link

Syntax:

html

```
<a href="URL">Link Text</a>
```

Example:

html

```
<a href="https://www.nytimes.com">The New York Times</a>
```

Result: User can click on "The New York Times" to go to the website

Adding Images

Image Tag ():

- Adds images to HTML page
- Has required attribute src (source) to give location of image
- Browser gets image and displays it on webpage
- Optional attributes: width, height to specify image size

Syntax:

html

```

```

Example:

html

```

```

Note: alt attribute provides alternative text for screen readers and when image fails to load

Creating Lists

Two Types of Lists:

List Type	Tag	Description
Unordered (Bulleted)		Items not in specific order
Ordered (Numbered)		Items in sequential order

List Item Tag: - used within both and

Nested Lists: Lists within lists

Example - Unordered List:

html

```
<ul>
  <li>Coffee</li>
  <li>Tea
    <ul>
      <li>Black tea</li>
      <li>Green tea</li>
    </ul>
  </li>
  <li>Milk</li>
</ul>
```

Example - Ordered List:

html

```
<ol>
  <li>First step</li>
  <li>Second step</li>
  <li>Third step</li>
</ol>
```

Creating Tables

Table Tags:

Tag	Purpose
<table>	Creates the table container
<tr>	Defines a table row
<td>	Defines a table cell (data)

Tag	Purpose
<th>	Defines a table header (bolded and centered)

Structure:

- <table> contains one or more <tr> tags (rows)
- Each <tr> contains many <td> tags (cells)
- <th> can be used instead of <td> for header cells

Example:

html

```
<table>
  <tr>
    <th>Rank</th>
    <th>Name</th>
    <th>Points</th>
  </tr>
  <tr>
    <td>1</td>
    <td>John</td>
    <td>150</td>
  </tr>
  <tr>
    <td>2</td>
    <td>Jane</td>
    <td>142</td>
  </tr>
</table>
```

Creating Forms

Definition: Forms are the principal mechanism to get user inputs on webpages. They allow users to interact with the webpage and submit information for processing.

Form Tag (<form>):

- Creates the form container
- Every form has multiple <input> tags to specify the kind of input desired
- Data collected in form is sent to server for processing using the action attribute

Syntax:

html

```
<form action="processor.php">
```

```
    <!-- form elements go here -->
```

```
</form>
```

Input Types:

Type	Description	Example
text	Single line text input	<input type="text">
password	Password field (masked)	<input type="password">
number	Numeric input	<input type="number">
email	Email address	<input type="email">
date	Date picker	<input type="date">
file	File upload	<input type="file">
radio	Radio button (select one)	<input type="radio">
checkbox	Checkbox (select multiple)	<input type="checkbox">
submit	Submit button	<input type="submit">

Textarea Tag (<textarea>):

- Multi-line input element
- Attributes: rows, cols to specify dimensions

Example:

html

```
<textarea rows="4" cols="50">
```

Enter text here...

```
</textarea>
```

Select Tag (Dropdown):

- Creates dropdown list
- `<option>` tags define available options

Example:

html

```
<select>
```

```
  <option value="volvo">Volvo</option>
```

```
  <option value="bmw">BMW</option>
```

```
  <option value="audi">Audi</option>
```

```
</select>
```

Radio Buttons:

- Allow user to select only one option
- All options in a group share the same name attribute

Example:

html

```
<input type="radio" name="gender" value="male"> Male
```

```
<input type="radio" name="gender" value="female"> Female
```

Checkboxes:

- Allow user to select one or more options

Example:

html

```
<input type="checkbox" name="hobby1" value="reading"> Reading
```

```
<input type="checkbox" name="hobby2" value="sports"> Sports
```

Complete Form Example:

html

```
<form action="/submit-form">

  <label for="fname">First Name:</label>

  <input type="text" id="fname" name="fname"><br>

  <label for="lname">Last Name:</label>

  <input type="text" id="lname" name="lname"><br>

  <input type="submit" value="Submit">

</form>
```

12.5 CASCADING STYLE SHEETS (CSS)

Definition: CSS stands for Cascading Style Sheets and is used to describe how HTML elements are to be displayed on screen, paper, phones, iPads, or in other media.

Purpose:

- Makes webpages stand out with color and design
- Saves developer time by allowing layout of multiple webpages to be controlled all at once
- Ensures consistency across entire website
- Separates content (HTML) from presentation (CSS)

Problem Without CSS:

- Adding and <color> attributes to HTML creates messy, inconsistent code
- Changing colors or font styles requires updating every element on every page
- HTML file becomes huge and messy

Solution:

- Define all styles in a CSS file
- Link CSS file to HTML documents
- Every element across all webpages will use the same styles

Three Ways to Apply CSS:

1. Inline CSS: Using style attribute directly in HTML tag

html

```
<p style="color:blue; font-size:16px;">This is blue text.</p>
```

2. Internal CSS: Using <style> tag in document head

html

```
<head>
```

```
  <style>
```

```
    p {
```

```
      color: blue;
```

```
      font-size: 16px;
```

```
    }
```

```
    h1 {
```

```
      color: navy;
```

```
      text-align: center;
```

```
    }
```

```
  </style>
```

```
</head>
```

3. External CSS: Separate CSS file linked to HTML

html

```
<head>
```

```
  <link rel="stylesheet" href="styles.css">
```

```
</head>
```

CSS File (styles.css):

css

```
body {
```

```
  background-color: lightgray;
```

```
  font-family: Arial, sans-serif;
```

```
}
```

```
h1 {
```

```
  color: navy;
```

```
  margin-left: 20px;
```

```
}
```

```
p {  
  color: black;  
  line-height: 1.5;  
}
```

Benefits of External CSS:

- Reduces clutter in HTML text
- One CSS file can control multiple HTML pages
- Changes made in one place affect entire website
- Improves maintainability and consistency

12.6 WEBSITE DESIGN AND STORYBOARDING

Website Definition:

Websites are collections of individual webpages that accomplish a specific purpose. When you visit a website, servers are waiting for you to reach out to them. Websites have a collection of webpages connected to one another through weblinks, providing a seamless experience.

Need for Planning:

- Building a website is resource-intensive
- Must plan, build, and deploy
- Should not build a website on the fly
- Start with storyboards

Storyboards

Definition: Storyboards are visual representations of creative projects. When used with websites, storyboards help all members of the development team identify all the users of a website and their expected interactions.

Origin: Evolved from film and drama; concept has found a home in website development

Purpose:

- Understand high-level overview and flow of webpages
- Visualize user experience
- Get team agreement about flow before proceeding

Wireframes

Definition: Wireframes are illustrations of the webpage that focus on space allocation for the elements of the page and their intended behaviors.

Characteristics:

- Intended to be simple and disposable
- Typically do not include any styling, color, or graphics
- Created during planning phase
- Help designers and developers identify and address potential design and usability issues

Content: Boxes and lines representing placement of different elements on the page:

- Headers
- Navigation menus
- Content sections
- Calls to action

Wireframing Tools:

- **Balsamiq:** Popular wireframing software; offers templates to pick, drag, and drop different webpage elements
- **Figma:** Another popular tool; acquired by Adobe in 2022 for \$20 billion; can automatically convert wireframe design into working HTML website

Process:

1. For large-scale projects, do storyboarding first
2. Move to wireframing tools
3. Build wireframes to refine application concepts
4. Test new application ideas with low-effort, no-cost method

Storyboard vs. Wireframe:

Aspect	Storyboard	Wireframe
Purpose	Storytelling and visual components	User interface and interaction aspects
Focus	User experience and flow	Space allocation and structure

Aspect	Storyboard	Wireframe
Detail	High-level overview	Specific placement
When used	First step	After storyboarding

12.7 STRUCTURE OF A WEBSITE

Folder Organization

Root Folder: The starting point of your application or website

Index.html: Default file displayed if someone visits your website and does not specify a specific file name

Recommended Folder Structure:

text

root/

|

|— index.html

|— about.html

|— contact.html

|

|— css/

| └— styles.css

|

|— js/

| └— script.js

|

|— images/

| └— logo.png

| └— banner.jpg

|

|— fonts/

 └— custom-font.woff

Benefits of Organized Structure:

- Easier to manage and maintain files
- Search engines can navigate website easily
- Users can navigate website easily

Content Management System (CMS):

- Provides graphical interface to organize and manage files and contents of websites
- Example: WordPress, Drupal, Joomla

Website Navigation Structure**Correspondence Between Menus and Folders:**

- Submenus of a website correspond to similarly named folders on the server
- Home page → root-level index.html
- Submenu 1 → Index.html within Submenu 1 folder

Example:

User visits home page → Server shows root-level index.html

User clicks Submenu 1 → Server shows /submenu1/index.html

12.8 HOSTING A WEBSITE

Definition: Hosting a website means making it accessible to the public over the Internet by storing the website's files and data on a webserver.

Webserver: A computer that is connected to the Internet and is designed to store and deliver webpages to users who request them through their browsers.

Process:

1. Rent space on a webserver from a hosting company or provider
2. Website is stored on the webserver
3. Made available for access over the Internet
4. Web hosting company provides range of services:
 - Server maintenance
 - Security
 - Technical support

Steps to Host:

1. Upload files to the webserver
2. Configure website settings
3. Acquire a domain name (such as www.example.com)
4. Point domain name to your website

Hosting from Your Home Computer:

- Get a permanent IP address from Internet service provider
- Purchase a domain name from a DNS registry
- Connect website name to computer's IP address
- Ensure computer can handle load, is protected, always turned on, and backed up regularly
- Adds a lot of complexity and responsibility

Popular Web Hosting Sites:

- GoDaddy
- Bluehost
- SiteGround
- HostGator
- DreamHost
- A2 Hosting
- InMotion Hosting
- Wix

12.9 WEBSITE EDITORS**WYSIWYG Editors**

Definition: WYSIWYG (Pronounced WIZ-EEE-WIG) stands for "What You See Is What You Get." In the context of HTML editors, a WYSIWYG editor is a software program that allows users to create and edit HTML documents without having to know the HTML code.

How It Works:

- Users drag and drop website elements (images, tables) on a drawing canvas
- See the webpage they are creating as it will appear in a browser

- No need to manually code the HTML

Advantages:

- Makes web technologies accessible to everyone
- Popular among non-technical users:
 - Bloggers
 - Content creators
 - Small business owners

Popular Free WYSIWYG Editors:

- **Brackets** from Adobe
- **Phoenix** code editor (from Brackets developer community)
- **SeaMonkey** from Mozilla community

Commercial WYSIWYG Editors:

- **Adobe Dreamweaver**

Website Builders:

- Website hosting providers like [Wix.com](https://www.wix.com) offer WYSIWYG platforms
- Drag and drop elements onto the page
- Click "Publish" button to deploy website
- No coding required

Google Sites (www.sites.google.com):

- Website hosting provider with its own WYSIWYG webpage builder

12.10 HTML EXTENSIONS**Dynamic HTML (DHTML)**

Definition: The collection of web technologies including HTML, CSS, and JavaScript to create interactive websites is called Dynamic HTML, or DHTML.

Division of Responsibilities:

Technology	Answers the Question
HTML	"What to display"
CSS	"How the display should look"
JavaScript	"How the webpage should interact with the user"

Capabilities:

- Create webpages that respond to user actions (mouse clicks, hover events)
- No need for page reload or server-side processing

Single Page Architecture (SPA):

- Development approach where content and functionality of a web application is contained within a single HTML page
- User remains on same page while sections get dynamically updated
- Initial HTML page loaded from server
- Subsequent interactions handled through JavaScript
- JavaScript code updates sections of the page without full page reload
- Offers seamless experience to visitors

Examples of SPAs:

- Gmail
- Facebook
- Twitter

Technology Behind SPA:

- **AJAX (Asynchronous JavaScript and XML):** Requests pieces of data from server and updates small sections on the page without disturbing user's flow

Criticism:

- **Doom-scrolling:** Users keep scrolling SPAs, anticipating interesting news
- Can be time sink and health hazard

Extensible Markup Language (XML)

Origin: HTML is designed specifically for publishing webpages. The popularity of HTML led to interest in generalizing this idea to standardize all forms of electronic publishing.

Definition: XML is the generalization of tags for electronic publishing. Several electronic publishing technologies use XML behind the scenes for their operation.

Key Feature: XML allows users to create their own custom tags to represent specific data structures and identify elements within a document.

Characteristics:

- Encodes documents in format that is both human-readable and machine-readable
- Can specify how every element of an electronic document should be interpreted
- Computers can precisely and easily transform information for different uses

XML Example:

xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<person>
```

```
  <name>John Smith</name>
```

```
  <age>30</age>
```

```
  <address>
```

```
    <street>123 Main St</street>
```

```
    <city>Anytown</city>
```

```
    <state>CA</state>
```

```
    <zip>12345</zip>
```

```
  </address>
```

```
</person>
```

Industry-Specific XML Standards:

- **XBRL (XML Business Reporting Language):** Helps organizations report financial performance in universal format
- Required by U.S. Securities and Exchange Commission for annual reports since 2018

Advantages of XML:

Advantage	Description
Human & Machine Readable	Easily read by humans and parsed/processed by computers
Platform Independent	Not tied to any specific platform or programming language
Extensible	Users can define their own tags for specific needs
Validation	Documents can be validated against defined schema or DTD
Data Exchange	Used for data exchanges between systems and applications
Separation of Content & Presentation	Clear separation between content and formatting

Document Type Definition (DTD):

- Specifies properties of each tag
- Defines whether elements are required or optional
- Computers can review every element and ensure data is properly formatted
- Helps avoid errors and inconsistencies in electronic documents

XML Reader: Software that reads an XML document and extracts the information contained within it, making it available for use in applications.

XML and JSON

Problem with XML - Verbosity:

- Of 28 words in the XML example, only 9 words (~33%) convey unique information
remaining 19 words (67%) are protocol tags
- Inefficient for information exchange

Solution - JavaScript Object Notation (JSON):

- Replaces tags with key-value pairs
- Eliminates closing tags
- Greatly reduces number of words needed to convey same information

JSON Example:

```
json
{
  "name": "John Smith",
  "age": 30,
  "address": {
    "street": "123 Main St",
    "city": "Anytown",
    "state": "CA",
    "zip": "12345"
  }
}
```

Comparison:

Aspect	XML	JSON
Syntax	Tags with opening/closing	Key-value pairs
Verbosity	High	Low
Readability	Good	Excellent
Data Types	Text only	Supports numbers, booleans, arrays
Parsing	Requires XML parser	Native JavaScript support
Use Cases	Highly regulated environments (financial reporting)	Most information exchange contexts

12.11 THE INTERNET AND WORLD WIDE WEB

Relationship Between Internet and Web

Concept	Definition
---------	------------

Internet	An interconnection of individual computers across the world through an interconnection of networks using a standardized set of communication protocols
-----------------	--

World Wide Web	An information system designed to display content in the form of documents, such as webpages which are stored on computers across the world
-----------------------	---

Analogy: The spiderweb design can represent the concept of the Internet, where computers are connected by specialized hardware and you can trace their communication in multiple directions.

HTTP and HTTPS:

- **HTTP (Hypertext Transfer Protocol):** An Internet protocol used by the World Wide Web that standardizes how documents such as webpages are transferred across systems
- **HTTPS:** Secure version of HTTP

Browser Navigation Bar: When you type `www.google.com`, your browser uses the HTTPS protocol to communicate the Google home page document content to the screen.

12.12 KEY TERMS AND DEFINITIONS

Term	Definition
------	------------

Anchor Tag	The <code><a></code> tag allows you to add links to an HTML page, also known as hyperlinks
-------------------	--

Bold Tag	The <code></code> tag is used to bold text
-----------------	---

Cascading Style Sheets (CSS)	A web markup standard that allows web designers to define the appearance and position of a webpage using special dynamic effects
-------------------------------------	--

DOCTYPE Declaration	The first line of an HTML page that tells the browser the kind of document to expect, specifically the version of HTML
----------------------------	--

Term

Definition

Form Tag

Created using the <form> tag; every form also has multiple <input> tags to specify the kind of input you want from the user

Head Tag

The head tag contains information about the HTML document; it is mostly not visible to the user

Heading Tag

The <h1> tag represents the main heading on a page and is typically used for the page title, while the other headings <h2> to <h6> are used for subheadings

HTML Tag

A tag is a special keyword that is enclosed by <>; the HTML tag defines the start and end of an HTML document

Hypertext Markup Language (HTML)

The standard markup language used for creating webpages and web applications

Image Tag

The tag is used to add images to your HTML page; it requires the source (src) attribute to give the location of the image

JavaScript

A programming language that lets you supercharge your HTML with animation, interactivity, and dynamic visual effects

Paragraph Tag

The <p> tag is used to define a block of text as a paragraph

POST

An HTML method that allows users to submit data to a server

Storyboard

A process used in the development of a website that involves the designer and colleagues simulating a particular set of visits to the site in order to gain ideas about how the site should be structured

Strike Tag

The <strike> tag is used to visually strikethrough text

Term	Definition
Subscript Tag	The <sub> tag is used to display text as subscript
Superscript Tag	The <sup> tag is used to display text as superscript
Tables	Created using the <table> tag, and <tr> and <td> tags to define the rows and cells of the table
Underline Tag	The <u> tag is used to underline text
Wireframe	A rough visual outline of a proposed application and is a specific kind of prototype generated during the user-centered design process to collect input and feedback from target users before designs are finalized
Webserver	Computer servers specifically designed and programmed to host websites and facilitate the communication between other computers across Internet networks
Website Host	Typically, a company or provider that allows users to maintain, secure, and support website domains on web servers

HTML Page Structure

text

```
<!DOCTYPE html>
```

```
<html>
```

```
  <head>
```

```
    <title>Page Title</title>
```

```
  </head>
```

```
  <body>
```

```
    <h1>Heading</h1>
```

```
    <p>Paragraph</p>
```

```
    <a href="url">Link</a>
```

```
    
```

```
  </body>
```

</html>

Essential HTML Tags

Purpose	Tag(s)
Headings	<h1> to <h6>
Paragraph	<p>
Bold	
Italic	<i>
Underline	<u>
Strikethrough	 or <strike>
Subscript	<sub>
Superscript	<sup>
Link	
Image	
Unordered List	+
Ordered List	+
Table	<table> + <tr> + <td>/<th>
Form	<form> + <input>

CSS Integration

- **Inline:** style="property:value;"
- **Internal:** <style> in <head>

- **External:** <link rel="stylesheet" href="file.css">

Website Development Process

1. **Storyboarding** - Visualize user experience and flow
2. **Wireframing** - Create simple layout mockups
3. **Development** - Code HTML, CSS, JavaScript
4. **Hosting** - Deploy to webserver
5. **Maintenance** - Update and improve

Computer Graphics

Chapter-10(10.1-10.4)

This chapter introduces the field of computer graphics, which is revolutionizing the creation of movies and interactive video games. Breakthroughs in computer graphics are breaking the bonds of real-world limitations in visual media, leading many to predict that computer-generated animation will soon eliminate the need for live actors, physical sets, and traditional photography across the film and television industries.

Computer graphics is the area of computer science that uses computing technology to create, generate, and modify visual content.

It covers a broad range of applications, such as:

- Displaying and formatting text
- Creating graphs, charts, and data visualizations
- Designing graphical user interfaces (GUIs)
- Editing and processing digital photographs and images
- Developing video games
- Producing animated films and motion pictures

In recent years, however, the term "**computer graphics**" is more and more commonly used to refer specifically to **3D graphics** (three-dimensional computer-generated imagery).

Because of this shift in common usage, most of this chapter focuses on **3D graphics** and explores its place within the larger field of computer graphics. We start by clearly defining what 3D graphics means and explaining how it fits into the overall discipline.

10.1 The Scope of Computer Graphics

The rise of digital cameras has led to a massive increase in the use of software designed to edit and manipulate digital images.

This type of software makes it easy to:

- Remove skin blemishes and fix "red-eye" effects in photographs
- Cut and paste elements from different photos to create completely new (and often unrealistic) images

Similar editing techniques are widely used in the motion picture and television industries to produce special effects. In fact, the ability to perform these kinds of manipulations was one of the main reasons these industries switched from traditional analog film to digital imaging systems.

Common special-effects applications include:

- Erasing visible support wires (e.g., for flying scenes)
- Layering or compositing multiple images together
- Generating short sequences of entirely new frames to change or enhance the originally captured action

Beyond photo and video editing, there is now a huge variety of utility and application software for creating two-dimensional (2D) images — from basic line drawings to professional digital artwork.

A simple and well-known example is Microsoft Paint, which lets users:

- Draw individual dots and lines
- Insert basic geometric shapes (circles, rectangles, etc.)
- Fill areas with solid colors
- Cut, copy, and paste selected parts of a drawing

All of these tools — photo editing, special effects, and 2D drawing programs — work with flat, two-dimensional content. They belong to two closely related fields of study:

- 2D Graphics Focuses on generating images by converting 2D shapes (such as circles, rectangles, text characters, etc.) into patterns of pixels.
- Image Processing Focuses on analyzing the pixels already present in an image to detect patterns, improve quality, enhance details, or even attempt to “understand” the content of the image. (We will study image processing later when we cover artificial intelligence topics.)

In short:

- 2D graphics = producing images from shapes
- Image processing = analyzing images made of pixels

In contrast, 3D graphics goes one step further by working with three-dimensional shapes and scenes.

The core idea of 3D graphics is:

- Build a digital model of a three-dimensional virtual scene (objects, lighting, materials, etc.)
- Then simulate a camera “photographing” that scene to produce a realistic 2D image

This process is very similar to traditional photography — except that the “scene” being photographed does not physically exist. It exists only as data and mathematical algorithms inside the computer.

Thus:

- Traditional photography → captures the real world

- 3D graphics → “photographs” virtual worlds

Creating a final image using 3D graphics involves two main stages:

1. **Modeling & scene creation** — Designing, encoding, storing, and editing the 3D virtual world (a creative and artistic process)
2. **Rendering** — Computing the 2D image from the 3D scene (a very computationally intensive process)

Because 3D graphics can create highly realistic images of imaginary scenes, it is perfect for:

- Interactive video games — The player explores and interacts with a fully encoded 3D virtual environment; every view is generated in real time using 3D graphics.
- Animated motion pictures — Animators work with the same kind of virtual 3D world, but a human director decides the camera angles and sequence. The final product delivered to audiences is a series of 2D rendered frames.

Looking to the future, as 3D graphics technology continues to improve and virtual worlds become more realistic and easier to create/manipulate, the way we experience movies could change dramatically.

Today, motion pictures are still distributed as sequences of 2D images — whether played from film reels (in the past) or digital files on DVDs, streaming, or flat-panel screens.

In the future, instead of distributing pre-rendered 2D footage, studios could distribute the complete 3D virtual world/set itself.

Viewers would receive access to the entire digital movie environment and use a 3D graphics “player” (similar to today’s game consoles) to experience it.

- By default, you could watch the “official” version — following the plot and camera choices intended by the director/producer.
- But you could also interact with the virtual set — exploring different angles, changing events, or creating your own alternate storylines — much like playing a video game.

The possibilities become even more exciting when combined with emerging three-dimensional human-machine interfaces (such as advanced VR/AR headsets, haptic feedback, gesture control, etc.).

In essence, 3D graphics is moving us from watching fixed 2D stories to experiencing living, interactive 3D virtual realities.

10.2 Overview of 3D Graphics

Overview of the 3D Graphics Process

Creating and displaying a 3D computer-generated image involves three main sequential steps:

1. **Modeling**
2. **Rendering**
3. **Displaying**

These steps form the core 3D graphics pipeline.

Step 1: Modeling

- Equivalent to building a movie set in traditional filmmaking
- In 3D graphics: “Constructing” a virtual 3D scene using digital data and mathematical algorithms
- The resulting scene never physically exists — it exists only as numbers (coordinates, shapes, materials, lights, etc.) inside the computer

Key points

- **Involves defining:**
 - **3D objects (geometry: vertices, edges, faces)**
 - **Materials (color, texture, shininess, transparency)**
 - **Lighting sources**
 - **Camera/viewpoint position**

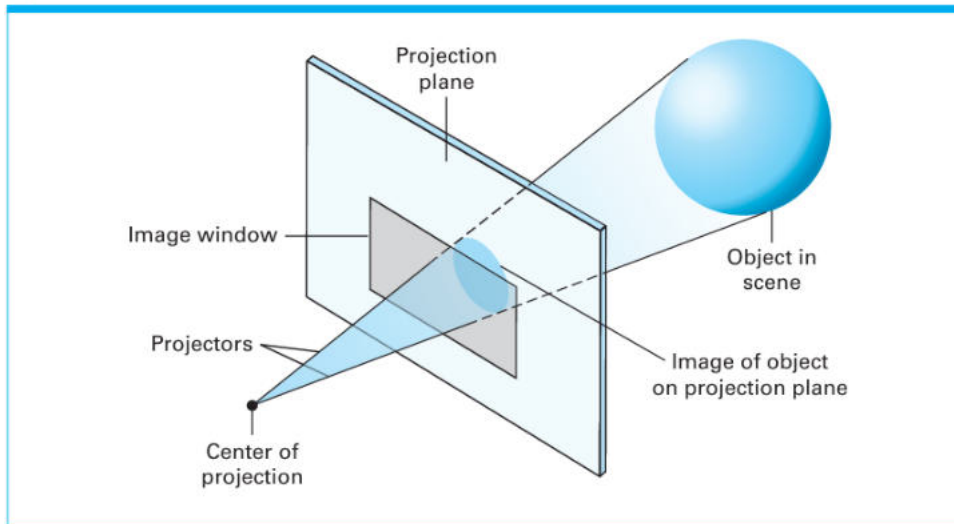
Step 2: Rendering

The process of computing a 2D image of the 3D virtual scene — as if a camera photographed it from a specific viewpoint which is shown in Figure 10.1

Core idea

- **Simulate how the scene would look if captured by a real camera**
- **Project 3D objects onto a flat 2D surface called the projection plane**

Figure 10.1 The 3D graphics paradigm



Types of Projection

Type	Projectors (lines of projection)	Appearance	Typical Use	Similarity to Human Vision
Perspective Projection	Converge to a single point (center of projection / viewpoint)	Objects farther away appear smaller; realistic depth	Most realistic images, video games, movies, human-eye-like view	Very close (what we see)
Parallel Projection	All projectors are parallel	No size reduction with distance; "true" shape & proportions	Engineering drawings, CAD, technical illustrations	Not realistic

Perspective Projection Details in 3D graphics

- All projection lines (called projectors) start from one point = center of projection (also called viewpoint or eye point)
- Produces foreshortening (distant objects look smaller) → mimics human eye / real camera

Important Terms

- Projection plane → Flat 2D surface where the image is formed (like film in a camera)
- Image window → Rectangular region on the projection plane that becomes the final image
 - Similar to the rectangle you see in a camera viewfinder

- Anything outside this window is cropped (e.g., Aunt Martha's head may be visible in the full viewfinder but cut off in the photo)

Pixel-by-Pixel Computation

- **For every pixel inside the image window:**
 - Determine which object(s) are visible at that pixel
 - Calculate how light interacts with the surface at that point
- **Factors that affect appearance (makes rendering complex):**
 - Material properties (shiny metal vs. soft fabric vs. glass)
 - Lighting (direct bright light vs. indirect/diffuse light)
 - Shadows (object blocked by another object)
 - Reflections (mirror-like surfaces showing other objects)
 - Transparency / refraction

Interdisciplinary Nature Rendering borrows heavily from:

- Analytic geometry (projection math)
- Physics (light behavior, optics)
- Material science (how surfaces reflect/absorb light)

Output of Rendering

- A bit map (pixel grid) of the final 2D image
- Stored in a special memory area called the frame buffer

4. Frame Buffer

Definition

- Memory area that holds the final pixel-by-pixel image (bit map)
- **Can be:**
 - Part of main RAM (software rendering)
 - Dedicated high-speed graphics memory (GPU/hardware-accelerated rendering)

Step 3: Displaying

- The contents of the frame buffer are sent to a display device (monitor, TV screen, VR headset, etc.) for viewing
- Or saved to permanent storage (file, video sequence) for later use

Two Main Contexts

Context	Real-Time Requirement	Typical Quality	Image	Reason for Quality Difference
Interactive applications (video games, flight simulators)	Must render & display in real time (30–120 fps)	Lower to medium (trade-off for speed)		Strict time deadline → simpler lighting, fewer details
Pre-rendered productions (animated movies)	No real-time constraint (can take hours/days per frame)	Very high (photorealistic)		Unlimited computation time → complex effects, global illumination, ray tracing

Typical Video Game System - Working

Components & Flow

1. Encoded virtual world → The game contains a complete 3D model (scene, objects, rules)
2. Player input → Player moves, shoots, explores → changes the virtual world
3. Game engine → Repeatedly:
 - Updates scene based on player actions
 - Performs rendering of the current view
 - Stores result in frame buffer
4. Special-purpose hardware (GPU) → Accelerates rendering (critical for real-time performance) → This is the main difference between a gaming console/PC with dedicated graphics card vs. a normal PC
5. Display device → Shows frame buffer contents → creates illusion of a moving, changing 3D world

7. 3D Graphics Pipeline

Step	Analogy (Traditional Film)	What Happens in 3D Graphics	Key Terms / Concepts	Output / Result
Modeling	Building physical movie set	Create 3D virtual objects,	Geometry, vertices, meshes, textures	Digital 3D scene description

Step	Analogy (Traditional Film)	What Happens in 3D Graphics	Key Terms / Concepts	Output / Result
		materials, lights, scene		
Rendering	Camera photographing the set	Project 3D scene → 2D image (perspective projection)	Viewpoint, projectors, projection plane, image window, frame buffer	2D pixel image (bit map)
Displaying	Showing the developed film	Send frame buffer to screen or save to file	Frame buffer, real-time vs. offline rendering	Visible image or stored video frame

8. Summary

3D graphics pipeline = Modeling → Rendering → Displaying

- Modeling → build virtual scene (never physically real)
- Rendering → compute 2D image using perspective projection (most realistic)
- Image window = final visible rectangle (like camera frame)
- Rendering is computationally expensive → needs physics, geometry, material science
- Frame buffer → holds final pixel image
- Real-time constraint → limits quality in games vs. movies
- GPU/hardware acceleration → essential for interactive 3D applications

Mnemonic “Model the scene → Render the view → Display the dream” (Scene doesn’t exist → Image is computed → Screen shows the illusion)

This forms the foundation of modern computer graphics — used in games, movies, VR/AR, simulations, and more.

10.3 Modeling

A 3D graphics scene is completely virtual — nothing physically exists; everything is digitally encoded.

Modeling in 3D Graphics:

Two main phases covered in this section:

1. Modeling individual objects (shape + surface properties)

2. Assembling objects into a scene

3. Factors Affecting Level of Detail in Object Modeling

The amount of detail in a model depends on:

Factor	More Detail Needed When...	Less Detail Acceptable When...
Position in scene	Object is in the foreground	Object is in the background
Real-time constraints	No strict time limit (e.g., pre-rendered movies)	Strict real-time requirement (e.g., video games)
Required image quality	High realism / photorealism desired	Approximate shape is sufficient

General Rule

- **More precise models → higher-quality images**
- **But also → longer rendering times**

Ongoing Research Goal Develop techniques for highly detailed yet efficient models, including:

- Models with multiple levels of detail (LOD) → Same object can switch between coarse (distant/low-priority) and fine (close/high-priority) versions dynamically

Information Needed to Describe an Object

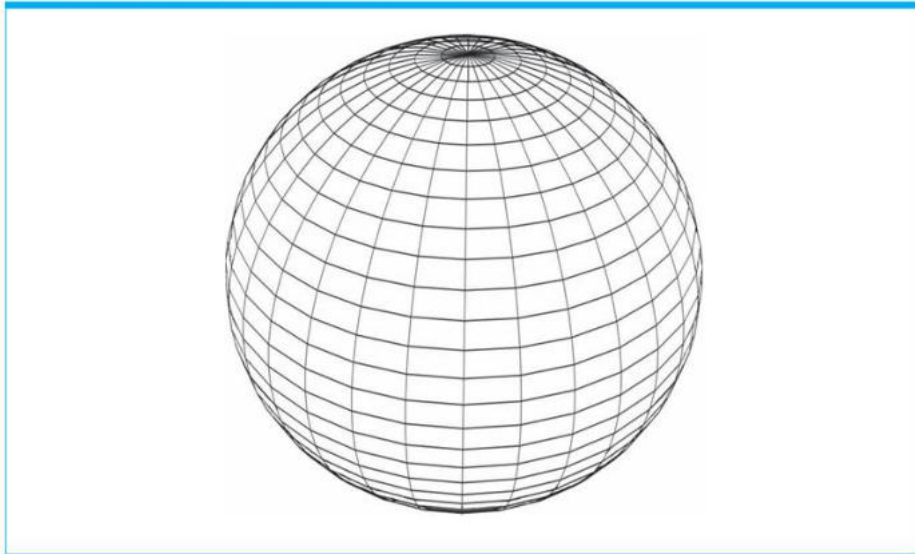
Two main categories:

1. **Shape (geometry) – discussed in detail below**
2. **Surface characteristics (material properties) – how the object interacts with light (color, shininess, texture, transparency, etc.) → Covered later in the chapter**

Modeling Shape: The Polygonal Mesh Approach

Dominant Method in Modern 3D Graphics Almost all objects are ultimately represented as a polygonal mesh – a collection of small flat surfaces called planar patches (usually polygons). A polygonal mesh for a sphere is as shown below:

Figure 10.2 A polygonal mesh for a sphere



Why Polygons?

- Easy to represent mathematically
- Uniform format → makes rendering entire scenes more efficient

Most Common Choice: Triangles

- Each triangle defined by only 3 vertices
- Minimum number of points needed to define a flat surface in 3D space
- Triangles are very efficient for hardware
- **Polygonal Mesh**
 - Collection of polygons (mostly triangles) connected edge-to-edge
 - Approximates the true shape of the object
 - More polygons (smaller patches) → smoother, more accurate approximation

Ways to Obtain a Polygonal Mesh

Method	Description	When Used / Examples	Advantages / Disadvantages
Geometric / Analytic Modeling	Start with precise mathematical equation → generate mesh points	Spheres, cylinders, cones, Bezier surfaces	Precise, elegant; but limited to simple shapes
Bezier Curves & Surfaces	Define smooth curves/surfaces using few control points	Car bodies, character models, organic shapes	Efficient for smooth curves; needs conversion to mesh

Method	Description	When Used / Examples	Advantages / Disadvantages
Manual Digitizing	Build physical model → use 3D digitizing pen to record surface points	Complex real-world objects (sculptures, artifacts)	Accurate for irregular shapes; slow & labor-intensive
Procedural Models	Write a program/algorithm that automatically generates the mesh	Trees, mountains, clouds, fire, smoke, water, crowds	Fast for complex, repetitive, or random structures

Examples of Geometric / Analytic Modeling

Sphere Example Equation (centered at origin): $r^2 = x^2 + y^2 + z^2$

- Generate lines of latitude & longitude
- Find intersection points → use as vertices
- Connect vertices to form triangular mesh

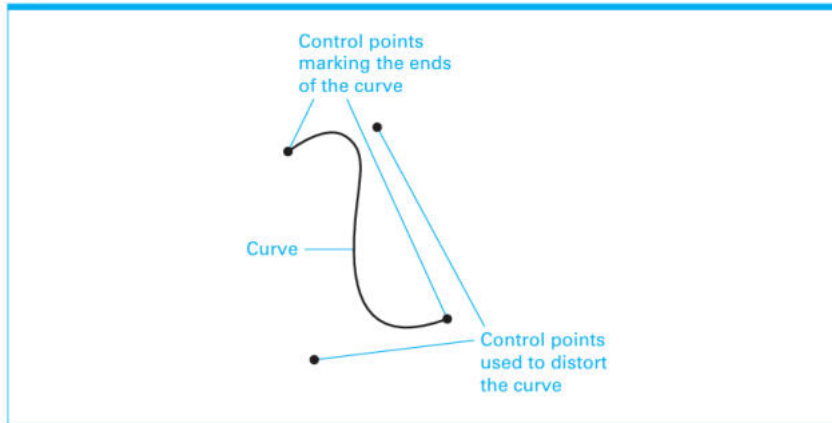
Result → Characters in low-budget animations often look “geometric” (made of spheres, cylinders, cones)

Bezier Curves

- Developed by Pierre Bézier (Renault, 1970s)
- A curve segment defined by control points (usually 4):
 - 2 endpoints
 - 2 intermediate points that “pull” the curve
- Moving control points changes shape dynamically
- Extended to Bezier surfaces for 3D modeling
- Often used as intermediate step → converted to polygonal mesh for rendering

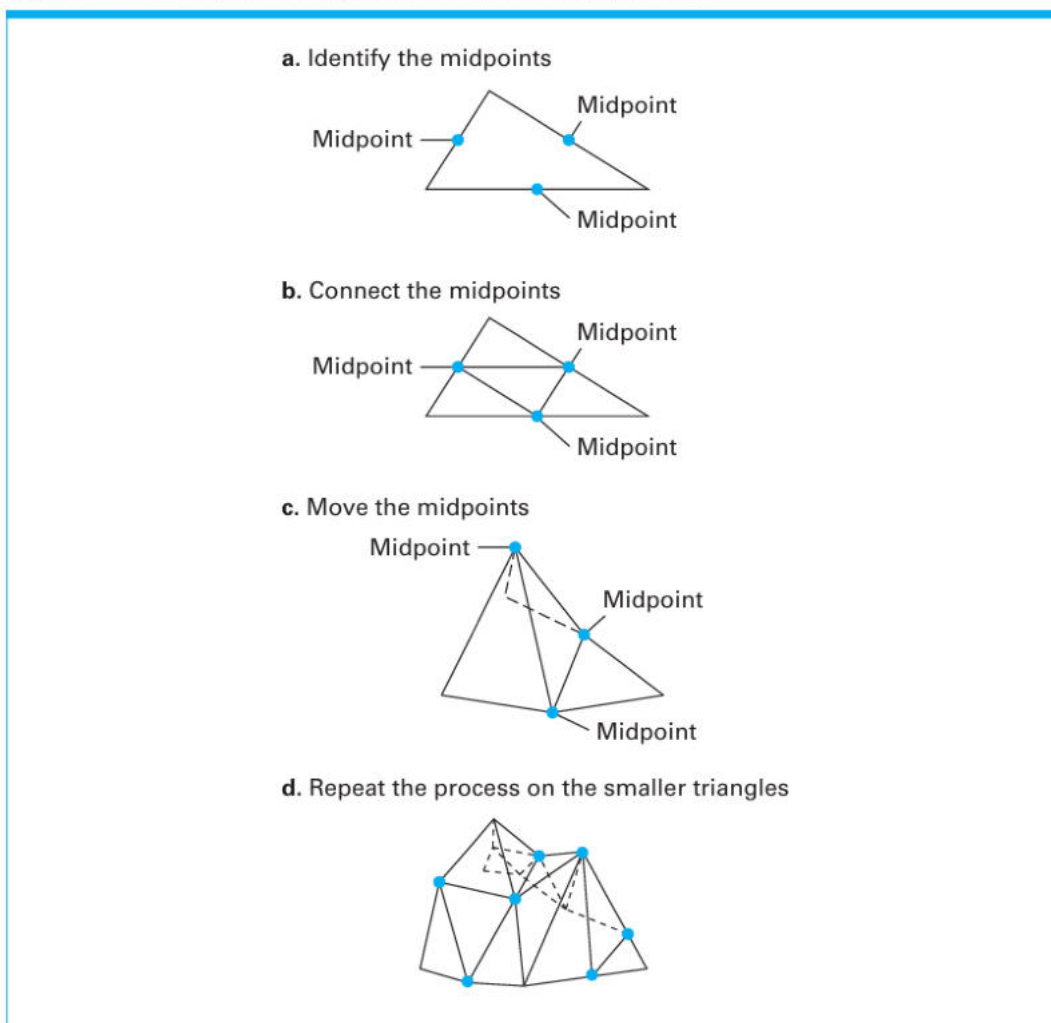
A Bezier curve is as shown below:

Figure 10.3 A Bezier curve



Procedural Models

Figure 10.4 Growing a polygonal mesh for a mountain range



Used when shapes are too complex for manual or simple geometric methods.

Examples of Procedural Techniques

Technique	How It Works	Typical Applications	Key Feature
Fractal Subdivision (e.g., mountains)	Start with triangle → split edges → displace midpoints → repeat recursively	Mountain ranges, terrain, rough surfaces	Creates natural-looking detail quickly
L-Systems Grammars /	Apply branching rules like a formal grammar (similar to parse trees)	Trees, plants, vascular structures	Each tree/plant is unique yet similar
Particle Systems	Large number of particles follow rules (physics-like behavior)	Fire, smoke, clouds, water sloshing, crowds	Simulates dynamic, fluid, or chaotic phenomena

Output Procedural models usually produce a polygonal mesh

Detail Level Control

- **Background tree** → coarse mesh (basic shape only)
- **Foreground tree** → fine mesh (individual branches & leaves)

Summary

3D scene = virtual set; objects = virtual props

- Modeling = constructing shape + surface properties
- Most objects represented as polygonal meshes (usually triangles)
- Trade-off: More detail → better quality but slower rendering
- Multiple Levels of Detail (LOD) = key research area
- Ways to build meshes: Geometric formulas, Bezier, digitizing, procedural
- Procedural models excel at complex/natural objects (trees, mountains, fire, etc.)
- Uniform mesh representation → enables efficient rendering of entire scenes

Surface Characteristics

- A polygonal mesh captures **only the geometric shape** (structure and form) of an object.
- It does **not** describe **surface appearance** – how the object looks under light (color, smoothness, roughness, texture, material behavior).
→ Rendering systems must receive **additional surface information** to produce realistic or desired visuals.

Surface Characteristics

Most rendering engines allow users to enrich a basic mesh during rendering:

Method	Description	Example Case	Use	Level of Control & Flexibility
Shading Techniques	Apply lighting models to simulate smooth, rough, shiny, matte surfaces	Render a ball as glossy red or rough green		High flexibility; user-defined look
Vertex Colors	Store color data directly at each vertex of the mesh	Reproduce scanned or painted colors on the object		Precise for original colors; limited patterns
Texture Mapping	Wrap a 2D image (texture map) onto the 3D surface like wallpaper	Add brick, wood, marble, fabric, or skin patterns		High realism with low polygons; very common

3. Texture Mapping – Core Concept & Working

Definition Texture mapping associates a **predefined 2D image** (the texture) with the surface of a 3D object.

Process

- During rendering, for any point on the 3D model, the system fetches the color from the **corresponding location** in the 2D texture image.
- Result: The low-polygon object appears to have detailed surface features.

Common Texture Examples

- Brick wall
- Wood grain
- Marble stone
- Fabric patterns
- Grass, metal, concrete, human skin details

Classic Example: Stone Wall

1. Create a simple polygonal mesh: long rectangular box (very few polygons).
2. Provide a 2D image of real stone masonry.
3. During rendering: apply the masonry image to the box. → Result: Realistic-looking stone wall with minimal geometry.

Advantages of Texture Mapping

- Extremely **efficient** – high visual detail with very low polygon count
- Ideal for **real-time applications** (video games, VR, interactive simulations)
- Widely used because it balances realism and performance

Limitations

- Best suited for **flat or gently curved surfaces**
- Heavy distortion occurs on strongly curved objects (e.g., texture stretched over a sphere looks unnatural – like wallpapering a beach ball)
- Visible **seams** can appear when the texture wraps around the entire object (pattern may not align perfectly at edges)

4. The Ongoing Search for Photorealism

Main Focus Developing object models and rendering techniques that produce **highly realistic images**, especially for **organic/living materials** associated with characters.

Key Research Areas & Challenges

Material	Major Challenges to Achieve Realism	Advanced Techniques Being Researched	Typical First Adoption
Human Skin	Subsurface scattering (light penetrates & scatters inside skin layers: epidermis + dermis)	Model light penetration depth, scattering behavior, layer contents (blood, melanin)	Motion pictures, high-end CG characters
Human Hair	~100,000 strands, translucency, color variation per strand, draping, response to wind/forces	Individual strand modeling, per-strand physics (dynamics, collision), scaled texture & color variation	Close-up shots in films & AAA games
Fur / Feathers	Dense layering, directional shadowing, volume appearance	Strand-based or volumetric fur rendering, shell texturing	Animal characters in animation
Cloth / Fabric	Weave/knit patterns (twill vs. satin vs. knit), yarn properties,	Simulate weave structure, thread-level mechanics, physics-based	Clothing in films, virtual fashion

Material	Major Challenges to Achieve Realism	Advanced Techniques Being Researched	Typical First Adoption
	stretching, shearing, draping	draping (stretch, shear, gravity)	

Key Insight Realism often requires:

- Fine **geometric detail** (individual strands/threads)
- **Physics simulation** (mechanical behavior, light interaction)
- Specialized **rendering algorithms** (subsurface scattering, strand dynamics)

NOTE

- Polygonal mesh → **shape only**
- Surface appearance added via: **shading, vertex colors, texture mapping**
- **Texture mapping** = fast, efficient, widely used in real-time graphics (games) → Best for flat surfaces; issues with curvature & seams

Modeling Entire Scenes

From Individual Objects to a Complete Scene

Once individual objects are modeled (shape + surface characteristics) and digitally encoded:

- Each object is assigned:
 - Location (position in 3D space)
 - Size (scaling)
 - Orientation (rotation)
- These positioned objects, along with other scene elements, are organized into a single data structure called a scene graph.

What is a Scene Graph?

Definition A scene graph is a hierarchical data structure that describes the entire virtual world (scene) for rendering.

Components Included in a Scene Graph

Element	Description	Analogy in Traditional Photography / Film
Objects / Props	Digitally encoded 3D models (with position, size, orientation)	Physical props and set pieces

Element	Description	Analogy in Traditional Photography / Film
Light Sources	Special objects representing lights (position, intensity, color, type)	Studio lights
Camera	Special object recording: position, orientation, focal properties (lens)	The camera / photographer's viewpoint
Background / Environment	Sky, ground, distant scenery (often simplified models)	Painted backdrops or set walls

- **Scene graph** $\approx$ complete studio setup in traditional photography
- **Contains everything needed to produce the final image: camera, lights, props, background**
- **Difference: Traditional setup** = physical objects; **Scene graph** = digitally encoded representations

Key Point The scene graph fully describes a virtual 3D world — nothing physical exists.

Importance of Camera Position

The camera's location and orientation in the scene graph has major effects:

Aspect	How Camera Position Influences It	Implication for Modeling
Foreground vs. Background	Determines which objects are close vs. far	Foreground needs high detail; background can be simpler
Level of Detail (LOD)	Detail requirement changes with distance from camera	Models must match expected visibility
Fixed vs. Dynamic Camera	Fixed camera (like a stage play) $\rightarrow$ roles are static	Model detail can be pre-decided
Moving / Interactive Camera	Camera moves (different views/"shots")	Detail may need to change dynamically

How (Virtual Reality)VR Uses Scene Graphs

- The imaginary 3D world is represented by a scene graph
- User movement $\rightarrow$ controls camera position & orientation
- To create depth perception (true 3D illusion):
 - Two cameras are used simultaneously:
 - One for right eye
 - One for left eye

- Each eye sees a slightly different image (stereoscopic rendering)
 - Images displayed separately to each eye (e.g., via VR headset)
- As an outcome User feels immersed inside the 3D virtual world
- Adding audio (spatial sound) + tactile/haptic feedback → illusion becomes very realistic

Pivotal Role of the Scene Graph in the 3D Graphics Pipeline

Position in the Workflow

Stage	Description	Nature of Work
Before Scene Graph	Artistic modeling: create objects, assign surfaces, textures, materials	Creative / artistic
Scene Graph Completion	All elements positioned, camera & lights set → virtual world fully defined	End of modeling phase
After Scene Graph	Rendering: compute projections, lighting, surface details, pixel colors	Computationally intensive / algorithmic

NOTE

- **Once the scene graph is built → modeling is finished**
- **Rendering becomes largely application-independent:**
 - **Compute projections**
 - **Determine visible surfaces**
 - **Simulate light interactions**
 - **Generate pixels**
- **Same rendering algorithms work for games, movies, simulations, VR, etc.**
- **Scene graph = organized collection of positioned objects, lights, and camera → describes virtual 3D world**
- **Analogous to a photography/film studio setup (but fully digital)**
- **Camera position decides foreground/background → affects required model detail**
- **Research: Adaptive / intelligent models that change detail as camera moves**
- **In VR: Two cameras → stereoscopic view → immersive 3D illusion**
- **Scene graph completion = end of modeling → start of rendering (computational phase)**

Rendering

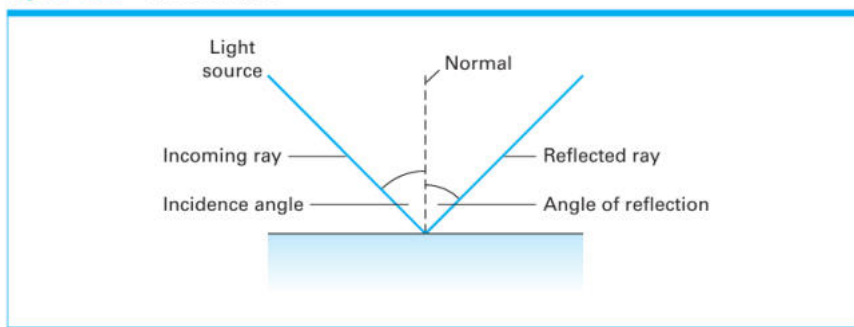
Rendering is the process of computing how objects in a scene graph appear when projected onto a 2D projection plane to form the final image. There are multiple rendering techniques, but this section focuses on the traditional approach. This traditional method is the most widely used in consumer graphics systems today. It powers popular applications such as video games, home computers, and mainstream graphics software. The next section will explore two alternative rendering approaches. The appearance of any object depends entirely on the light it emits or reflects toward the viewer/camera. Therefore, rendering fundamentally involves simulating the behavior of light as it interacts with objects. Accurate light simulation determines realistic colors, shadows, highlights, and surface details. The goal is to produce a believable 2D image that matches what a real camera (or eye) would see. Understanding light-object interaction forms the essential foundation for all rendering techniques.

Light-Surface Interaction

When light strikes an object's surface, it can be absorbed, reflected, or refracted (bent and passed through), depending on the material.

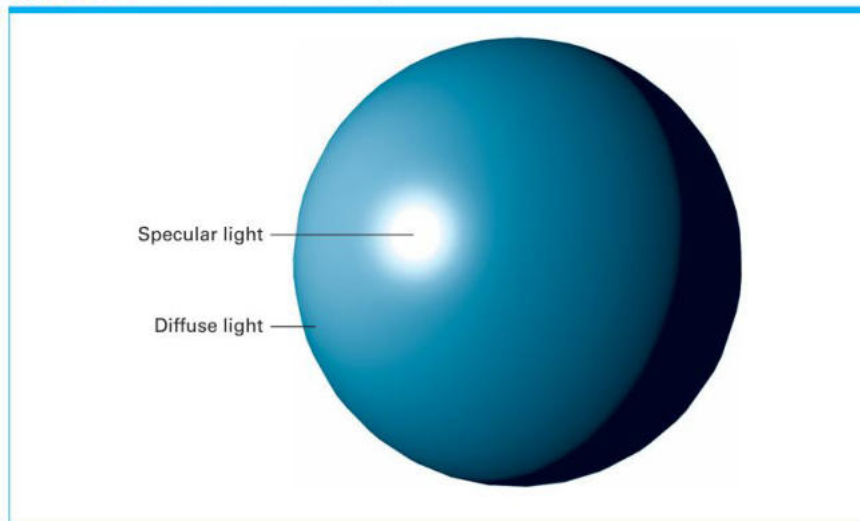
Reflection

Figure 10.5 Reflected light



For reflection on an opaque flat surface, the incoming ray hits at the angle of incidence (measured from the normal). The reflected ray leaves at the same angle, on the opposite side of the normal, and all three (incoming ray, reflected ray, normal) lie in one plane. If the surface is perfectly smooth, parallel incoming rays reflect as parallel rays → this is called specular reflection (shiny highlights). Specular light is visible only when reflected directly toward the viewer and usually shows the color of the light source. Real surfaces are rarely perfectly smooth, so many rays scatter in different directions after interacting with surface irregularities → this is diffuse reflection. Diffuse light spreads widely, is visible from many angles, and carries more of the object's own color due to longer surface contact. In a typical scene (e.g., a ball under one light), the bright spot is specular reflection, while the illuminated side shows diffuse light. The dark side (away from the light) is visible thanks to ambient light — scattered, directionless stray light from the environment. Most surfaces produce a mix of specular and diffuse light; smooth surfaces emphasize specular (shiny), rough surfaces emphasize diffuse (dull). Some surfaces are anisotropic: their reflection ratio (specular vs. diffuse) changes with viewing angle or light direction → appearance shifts from shiny to dull when rotated.

Figure 10.6 Specular versus diffuse light

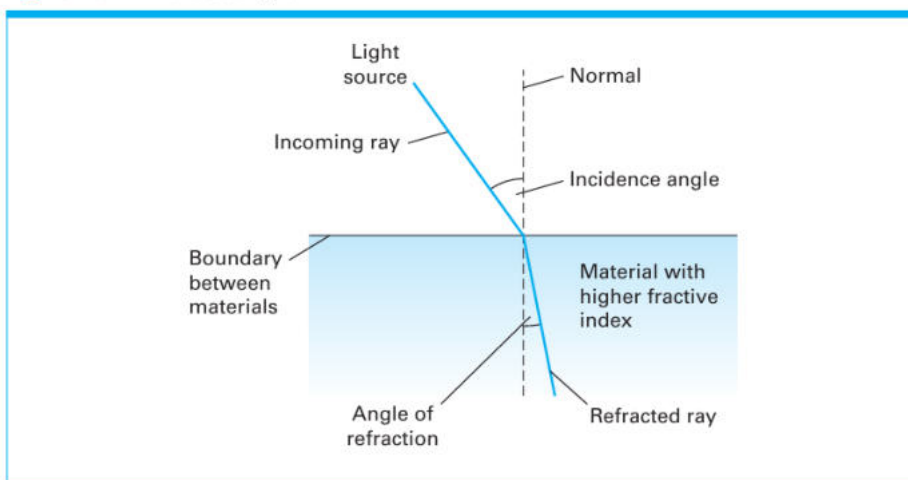


Examples of anisotropic surfaces include satin fabric (nap direction) and mowed grass fields (striped light-dark patterns due to grass lie).

Refraction

When light strikes a transparent object (unlike an opaque one), the rays pass through the surface rather than reflecting off it.

Figure 10.7 Refracted light



As the light enters the material, its direction changes — this bending is called refraction.

The amount of bending depends on the refractive index of the materials involved.

The refractive index is linked to the material's density: denser materials generally have a higher refractive index.

- When light moves from a lower-index material (e.g., air) to a higher-index one (e.g., water or glass), it bends toward the normal (the perpendicular line at the surface).
- When moving from higher to lower index, it bends away from the normal.

To correctly render transparent objects, rendering software must know:

- The refractive indexes of both materials at the boundary (e.g., air-glass, glass-water).
- The orientation of the surface: which side is the outside and which is the inside of the object (i.e., is light entering or exiting?).

Determining inside vs. outside can be subtle.

One common technique: Always list the vertices of each polygon in a counter-clockwise order when viewed from outside the object.

This ordering convention automatically encodes surface orientation, allowing the renderer to correctly decide whether light is entering or exiting the material.

NOTE: Refraction simulates realistic bending of light through transparent objects like glass, water, or crystals, requiring both refractive index data and proper surface normal/orientation information.

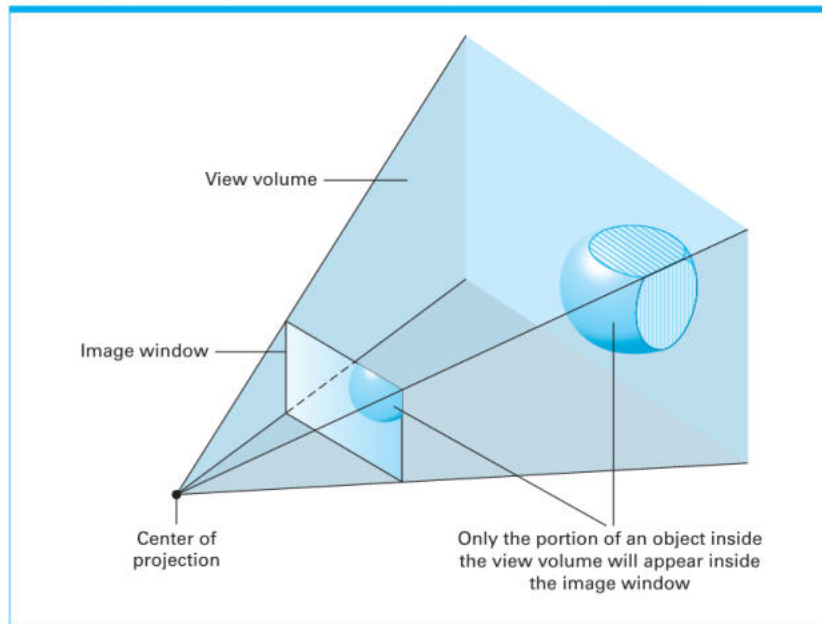
Clipping, Scan Conversion, and Hidden-Surface Removal

- The rendering pipeline is the standard process used in most interactive video game systems and consumer graphics applications.
- It focuses on opaque objects → ignores refraction, mirrors, shadows, and inter-object light interactions (for simplicity).
- Goal: Convert a 3D scene graph into a 2D image in the frame buffer by projecting visible parts onto the projection plane.

Step 1: Identifying the View Volume

- The view volume is the 3D pyramid-shaped region visible to the camera.
- It is defined by straight lines (projectors) from the center of projection (camera viewpoint) through the corners of the image window which is shown in below Figure.

Figure 10.8 Identifying the region of the scene that lies inside the view volume



- Only objects (or parts of objects) inside this volume will appear in the final image.

Step 2: Discarding Objects Outside the View Volume

- Quickly eliminate entire objects that are completely outside the view volume.
- Optimization: Scene graphs are often organized as trees → large branches (groups of objects in distant regions) can be discarded entirely.
- This step saves a lot of computation.

Step 3: Clipping (Trimming Objects)

- Clipping removes the parts of objects that lie outside the view volume.
- Performed on each planar patch (polygon/triangle) individually.
- Result: Only polygonal meshes (or trimmed patches) that lie entirely inside the view volume remain.
- Analogy: Slicing off the parts of a scene that fall outside the camera's field of view.

Step 4: Scan Conversion (Rasterization)

- Goal: Map points on the remaining planar patches to pixel positions in the final image.
- Only points that land exactly on pixel centers contribute to the image.
- Small details between pixels are lost → higher resolution (more pixels) captures finer details (reason digital cameras advertise megapixels).

How Scan Conversion Works

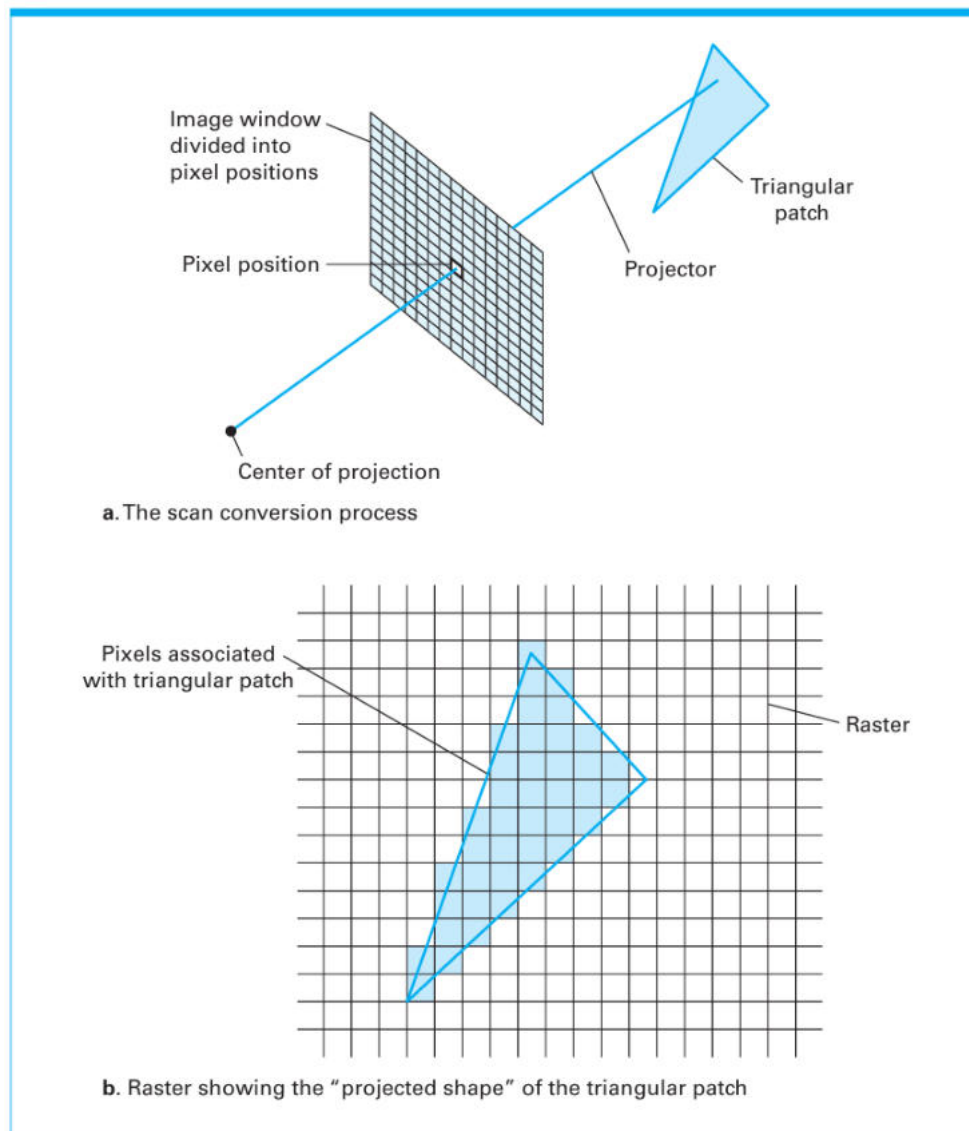
- Extend a projector (straight line) from the camera through each pixel center in the image window.

- Find where this projector intersects the planar patches.
- The intersection point determines the pixel's color and appearance.

NOTE:

- **Scan conversion = converting shapes into horizontal rows of pixels (scan lines)**
- **Rasterization = filling the pixel grid (raster) with appropriate colors**
- **Jagged edges ("jaggies") occur when features are smaller than pixel size (common on low-resolution displays).**

Figure 10.9 The scan conversion of a triangular patch



Step 5: Hidden-Surface Removal

Problem Multiple patches may intersect the same projector → only the closest (nearest to the camera) should be visible.

Simple Technique: Back-Face Elimination

- Discard polygons whose normal faces away from the camera (the back side of objects).
- Fast and easy → eliminates about half the polygons in closed objects.

Complete Hidden-Surface Removal

- Needed when objects overlap (e.g., car in front of a building).
- Only the foreground object should be rendered at each pixel.

Painter's Algorithm

- Sort objects by distance from camera (farthest first).
- Render distant objects first → closer ones overwrite them.
- Fails when objects interpenetrate (e.g., tree branches behind and in front of another object).

Z-Buffer (Depth Buffer)

- Extra memory buffer (same size as frame buffer) storing depth (distance from camera) for each pixel.
- Initialization: Set every z-buffer entry to maximum possible distance.

Z-Buffer Algorithm (Step-by-Step)

1. For each point on a planar patch being considered:
2. Calculate its distance from the camera along the projector.
3. Compare this distance to the current value in the z-buffer for that pixel.
4. If new distance is closer (smaller value):
 - Compute the point's appearance (color, shading, texture).
 - Update the frame buffer with this color.
 - Update the z-buffer with the new (closer) distance.
5. If new distance is farther → ignore (point is hidden or already blocked).

Advantages of Z-Buffer

- Works regardless of object order.
- Handles complex overlapping scenes easily.
- Very widely used in real-time graphics (games, GPUs).

NOTE

- Rendering pipeline = standard for real-time games → handles opaque objects only.
- View volume → camera's "visible pyramid".
- Clipping → cuts off outside parts.

- Scan conversion / rasterization → turns 3D patches into pixels.
- Hidden-surface removal → z-buffer is the dominant method (depth testing).
- No shadows, reflections, refraction in basic pipeline → keeps it fast for real-time.

Shading

After scan conversion identifies a point on a planar patch that maps to a pixel in the final image, the next task is to determine how that point should look — this process is called shading.

- Shading computes the characteristics of light reaching the camera from that point.
- The key factor is the orientation (tilt/angle) of the surface at that exact point.
- Surface orientation determines how much specular, diffuse, and ambient light is visible to the camera.

Flat Shading (Simplest Method)

- Assumption: The entire planar patch is perfectly flat → use the same orientation (normal vector) for every point on the patch.
- Result: Each patch is shaded uniformly → final image looks faceted (blocky, polygonal appearance) rather than smooth and rounded.

Figure 10.10 A sphere as it might appear when rendered by flat shading



- Essentially shows the polygonal mesh structure itself, not the smooth object it approximates.
- Fast and simple, but produces unrealistic “faceted” look (e.g., a sphere looks like a faceted gem).

Goal of Advanced Shading: Smooth, Realistic Appearance

To make curved objects look smooth (even though modeled with flat polygons), the renderer must estimate the true surface orientation at each rendered point inside the patch. → This creates a smoothly blended, curved-looking surface from flat polygonal patches.

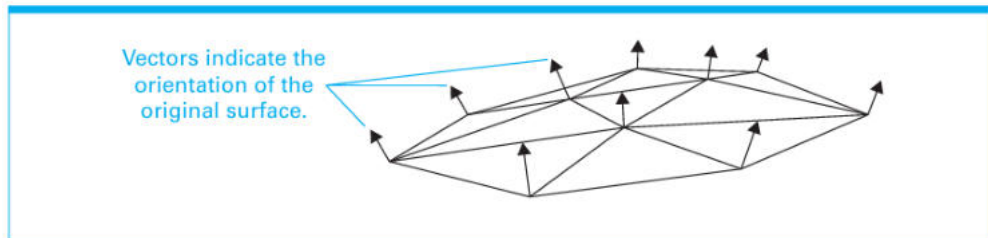
How to Estimate Surface Orientation

Most techniques start by knowing the surface orientation (normal vector) at the vertices of the polygonal mesh.

Ways to Obtain Vertex Normals

- During modeling: Explicitly store the true normal vector of the original smooth surface at each vertex (best accuracy).
- Computed automatically: Average the normals of all patches that share the vertex. Result: Each vertex has an attached normal vector pointing outward, perpendicular to the original smooth surface (see Figure 10.11).

Figure 10.11 A conceptual view of a polygonal mesh with normal vectors at its vertices



Main Shading Strategies Using Vertex Normals

Technique	What It Interpolates	Order of Operations	Strength / Weakness	Best For
Gouraud Shading	Interpolates color (appearance)	1. Compute color at vertices 2. Interpolate colors across patch interior	Fast; good for diffuse lighting; misses sharp specular highlights inside patches	Diffuse surfaces, real-time speed
Phong Shading	Interpolates normal vectors (orientation)	1. Interpolate normals across patch 2. Compute color using interpolated normal	More accurate specular highlights inside patches; detects small orientation changes	Shiny / specular surfaces

Key Difference

- **Gouraud:** Converts orientation → color at vertices → interpolates color.
- **Phong:** Interpolates orientation → computes color at each point. → Phong is better at capturing specular highlights that appear in the middle of a patch (because it responds to subtle orientation changes).

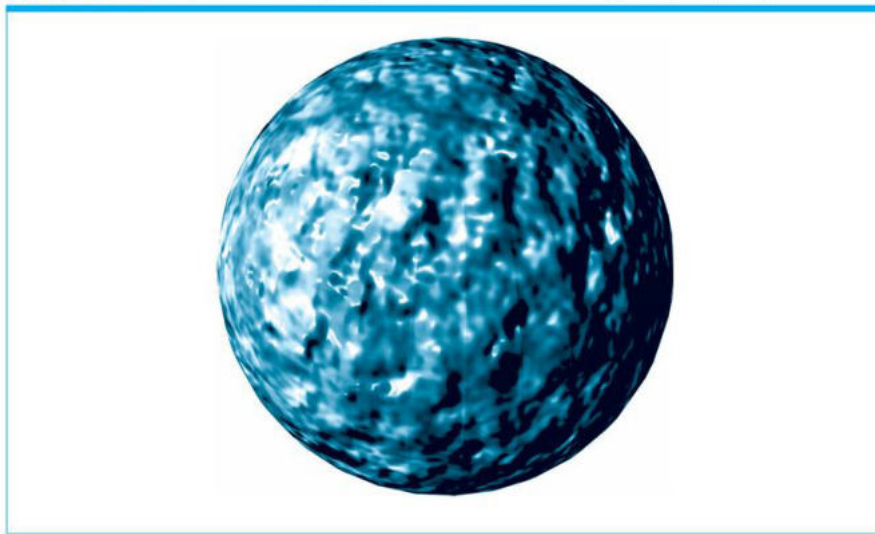
Enhancing Shading with Texture & Bump Mapping

Basic shading can be extended to simulate surface texture and roughness without adding extra geometry.

Bump Mapping

- A technique that adds small random variations to the interpolated surface normals.
- Makes the surface appear rough or bumpy (e.g., orange peel, leather, rough stone) without changing the actual polygonal mesh.
- Works by perturbing normals during shading → light interaction creates illusion of depth and texture.
- Result: Low-polygon model looks much more detailed and realistic (see Figure 10.12).

Figure 10.12 A sphere as it might appear when rendered using bump mapping



NOTE:

- Shading = determining light/color at each pixel based on surface orientation.
- Flat shading → fast but faceted (shows polygons).
- Gouraud shading → interpolates colors → smooth but misses interior specular highlights.

- **Phong shading** → interpolates normals → better specular highlights → more realistic for shiny surfaces.
- **Vertex normals are critical** → obtained from modeling or averaging adjacent faces.
- **Bump mapping** → simulates roughness/texture by perturbing normals → huge realism boost with low polygon count.

Rendering-Pipeline Hardware

The rendering pipeline (clipping → scan conversion/rasterization → hidden-surface removal → shading) is a well-established sequence for producing 2D images from 3D scenes.

These operations have been implemented directly in specialized hardware using VLSI technology, resulting in powerful chips (GPUs) that automatically handle the entire pipeline.

Modern low-cost GPUs can render millions of polygons per second, making real-time graphics feasible.

Dedicated graphics systems (e.g., video game consoles) include this hardware by default. In general-purpose computers, it is added via a graphics card (graphics adapter) connected to the system bus.

Advantages

- Dramatically speeds up rendering → enables high frame rates.
- Greatly simplifies application software: the program only needs to supply the scene graph to the hardware.
- The hardware performs all pipeline steps and writes the result directly to the frame buffer for display.
- From the software viewpoint, the complex pipeline is reduced to one abstract step (send scene → hardware renders).

Example: Video Game Workflow

- At startup: Game software sends the scene graph to the GPU.
- GPU renders the initial frame → stores it in the frame buffer → displayed on screen.
- During gameplay: Software updates the scene graph (player actions, object movement).
- GPU re-renders the updated scene every frame → continuous smooth animation.

Challenge: Hardware Differences Different GPUs have varying capabilities and command formats → a game built for one platform needs modification to run on another.

Solution: Standard Graphics APIs Software interfaces act as intermediaries, translating standardized commands into hardware-specific instructions. Examples:

- OpenGL (Open Graphics Library) – non-proprietary, widely used in games and cross-platform.

- Direct3D (part of DirectX) – Microsoft's API, standard for Windows environments.

Main Limitation The pipeline uses a local lighting model → each object is lit independently, ignoring interactions with other objects (e.g., no real shadows cast by one object on another, no reflections).

In contrast, global lighting models consider inter-object light effects but are too slow for real-time use with current technology.

Workarounds for Some Global Effects Fake drop shadows (shadows on the ground) can be simulated within the local model:

- Duplicate the object's mesh.
- Flatten/squash the copy.
- Place it on the ground plane.
- Color it dark.
- Render it normally → illusion of a shadow.

This technique is widely used in both consumer games and professional applications like flight simulators.

NOTE: Rendering-pipeline hardware (GPUs + APIs like OpenGL/Direct3D) enables fast, real-time 3D graphics by offloading complex calculations from software to dedicated silicon, but sacrifices true global illumination for speed. Fake techniques bridge some gaps until global methods become real-time capable.