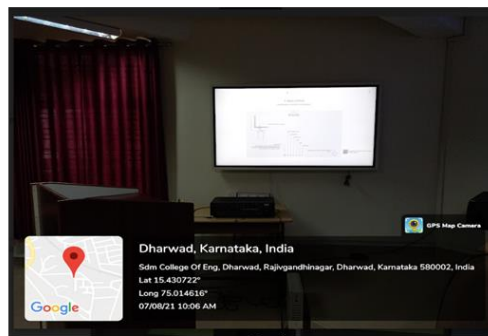
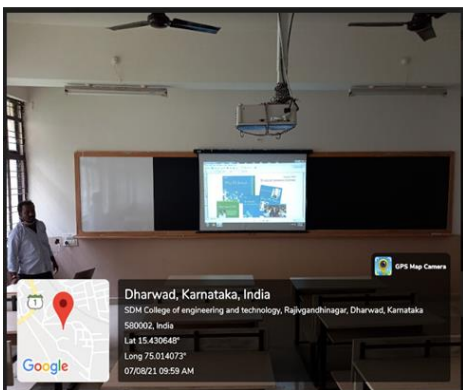
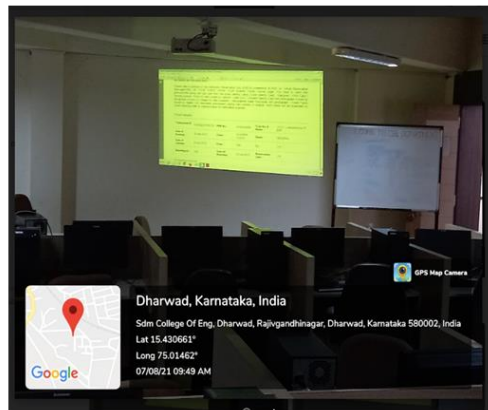
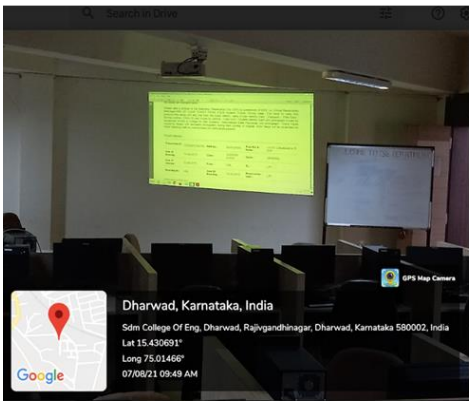
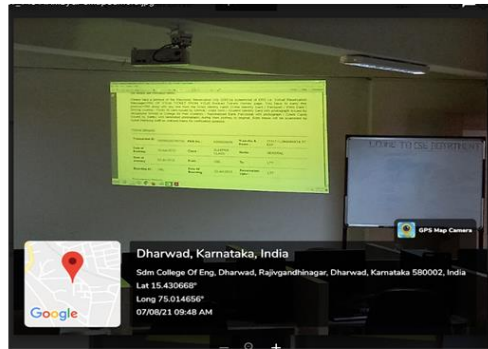
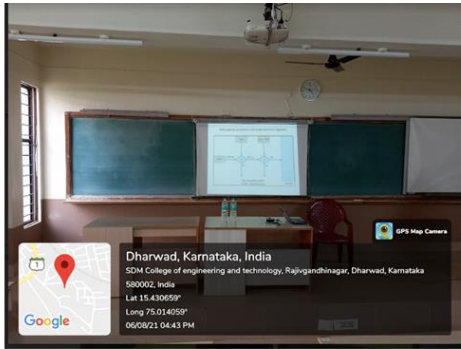
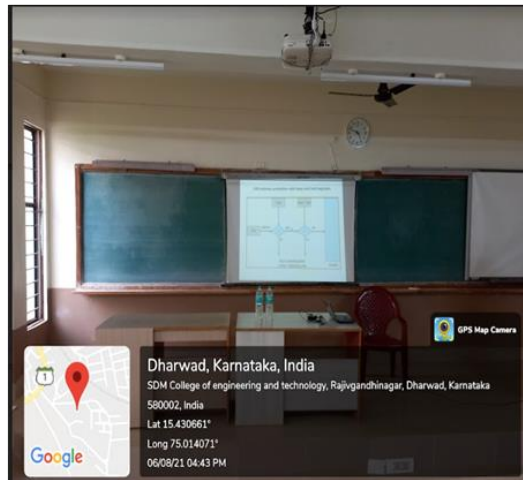
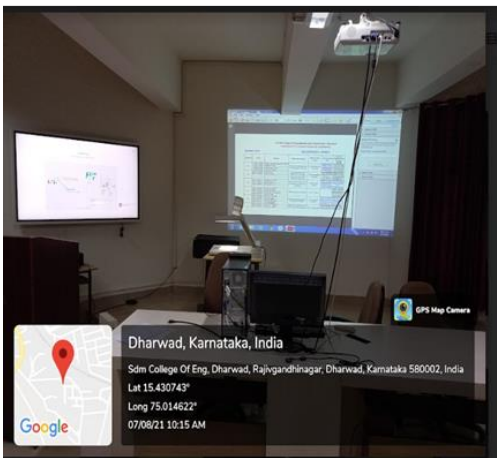
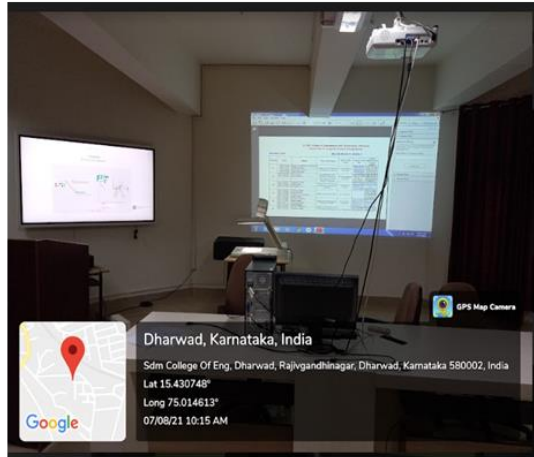
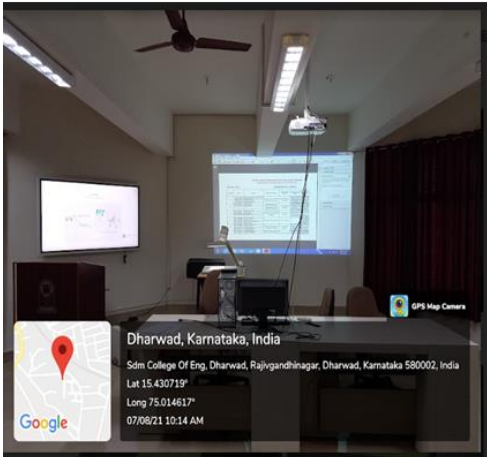
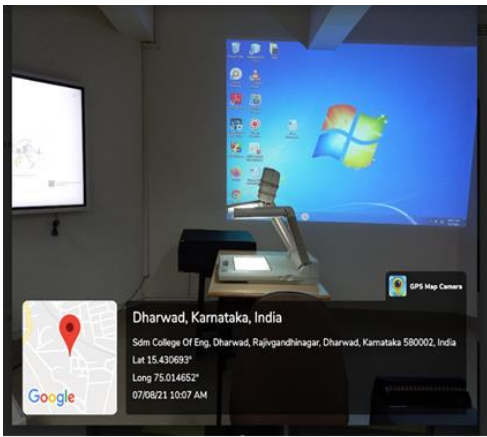
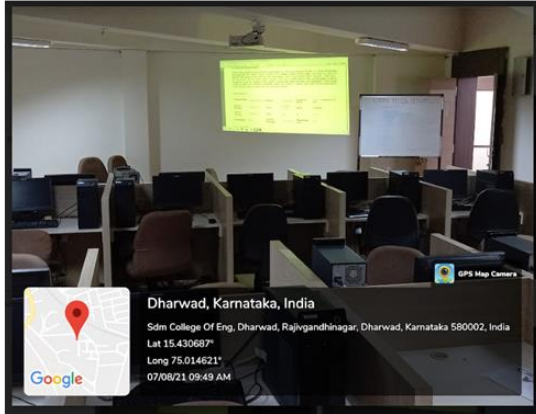
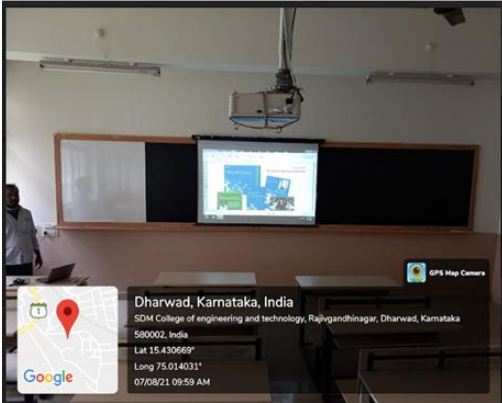
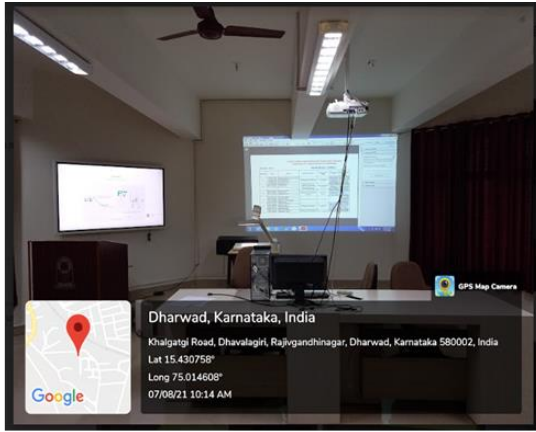


2.3.2: Teachers use ICT enabled tools including online resources for effective teaching and learning process

1) ICT supported Classrooms:







2) Screenshots of sample PowerPoint presentations:

1.

The first set of screenshots (1-12) shows a PowerPoint presentation titled 'Unit-1: GUI Programming with JavaFX'. The slides include:

- Slide 1:** Unit-1: GUI Programming with JavaFX. Includes course details: 18ACE353B - ADVANCED OBJECT ORIENTED PROGRAMMING, 5TH SEMESTER & DIVISION ACCOUNTS YEAR: 2021 - 22, DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, SOMJET, DHARWAD-2.
- Slide 2:** Introducing JavaFX GUI Programming. CHAPTER-34: JAVA THE COMPLETE REFERENCE NINTH EDITION, HERBERT SCHLÖTT.
- Slide 3:** Preamble. Lists reasons for JavaFX development: Java continues to evolve, original GUI framework (AWT/Swing) is outdated, Swing is not a native approach, JavaFX is designed for modern consumer applications.
- Slide 4:** Preamble (cont...). Lists features: Consumer applications demand a GUI that is visually capable, current trend is towards more reactive visual effects, JavaFX is Java's next-generation client platform and GUI framework, provides powerful, streamlined, flexible framework, simplifies creation of modern, visually exciting GUIs.
- Slide 5:** JavaFX vs. AWT & Swing. Compares AWT, Swing, and JavaFX examples.
- Slide 6:** History. Lists milestones: JavaFX development in two phases, original JavaFX based on scripting language called `javafx:script`, discontinued with JavaFX 2.0, supported FXML, bundled with Java since JRE7 Update 4, supports JavaFX 8, lightweight and supports MVC architecture, facilitates a more visually dynamic approach to GUIs.
- Slide 7:** JavaFX Basic Concepts. Explains rendering, packages, and classes like `javafx.application`, `javafx.stage`, `javafx.scene`, and `javafx.scene.layout`.
- Slide 8:** Stage and Scene Classes. Defines container relationships: Stage contains Scene, Scene contains Scene Graph, Scene Graph contains Nodes.
- Slide 9:** Stage and Scene Classes (cont...). Details Stage as a top-level container and how it handles scene updates.
- Slide 10:** Nodes and Scene Graphs. Defines nodes, child nodes, parent nodes, and scene graphs.
- Slide 11:** Nodes and Scene Graphs (cont...). Explains root nodes, parent-child relationships, and Node subclasses.
- Slide 12:** JavaFX Application Layers. Shows a diagram of the application layers: Stage, Scene, Scene Graph, and Nodes.

2.

The second set of screenshots (1-12) shows a PowerPoint presentation titled 'Introduction to IoT'. The slides include:

- Slide 1:** Introduction to IoT. Includes a book cover for 'Internet of Things: A Basic-on approach' by Harchandee Bahga, Vijay Madisetti.
- Slide 2:** Introduction. A cartoon illustration of a person interacting with smart devices.
- Slide 3:** Introduction. A cartoon illustration of a person in a smart home environment.
- Slide 4:** OUTLINE. Lists topics: IoT Definition, Characteristics of IoT, Physical Design of IoT, Logical Design of IoT, IoT Protocols, IoT-Infra and Deployment Templates.
- Slide 5:** WHAT IS IOT?. Defines IoT as a network of physical objects connected to the internet, enabling data exchange and remote control.
- Slide 6:** INTRODUCTION. Shows a diagram of IoT applications in various sectors like healthcare, agriculture, and industry.
- Slide 7:** INTRODUCTION. Explains that IoT comprises things with unique identities and connections to the internet, used for configuration, control, and networking.
- Slide 8:** INTRODUCTION. Lists drivers for IoT: Smart Devices, Mobile Devices, Wireless Communications, Sensing, Cloud Technologies. Includes a bar chart showing IoT market growth from 2010 to 2015.
- Slide 9:** INTRODUCTION. Explains the scope of IoT, from connecting things to the internet to enabling meaningful applications.
- Slide 10:** INTRODUCTION. Shows a diagram illustrating the flow from Data to Information to Knowledge.
- Slide 11:** INTRODUCTION. Shows a diagram illustrating the application of IoT in various domains.
- Slide 12:** DEFINITION OF IOT. Defines IoT as a dynamic global network infrastructure with self-configuring capabilities based on standard and interoperable communication protocols.



3.

Lighting and Shading

Objectives

- Learn to shade objects so their images appear three-dimensional
- Introduce the types of light-material interactions
- Build a simple reflection model the Phong model that can be used with real time graphics hardware

Why we need shading

- Suppose we build a model of a sphere using many polygons and color it with $\text{Color}(R, G, B)$, we get something like:
- But we want:

Shading

- Why does the image of a real sphere look like:
- Light-material interactions cause each point to have a different color or shade
- We need to consider:
 - Light sources
 - Material properties
 - Location of observer
 - Surface orientation

Meanings of words

- Opaque**: Allowing little light to pass through
- Translucent**: Allowing light to pass through, but diffusing it
- Transparent**: Allowing light to pass through almost unobstructed
- Diffuse**: Scattered over through
- Specular**: Pertaining to mirrors or mirror-like

Scattering

- Light strikes A
- Some absorbed
- Some scattered
- Some of scattered light strikes B
- Some absorbed
- Some of scattered light strikes A and so on
- This recursive nature accounts for subtle shading effects
- Can model such scenario using an integral equation (rendering equation)
- Used to find shading of all surfaces in a scene

Rendering Equation

- The infinite scattering and absorption of light can be described by the rendering equation
- Cannot be solved analytically
- Numerical methods for computing a solution are not fast enough for real-time rendering
- Can use approximate approaches like radiosity and ray-tracing
- Ray tracing is a special case for perfectly reflecting surfaces
- Rendering Equation is global and includes:
 - Shadows
 - Multiple scattering from object to object

Global Effects

- Consider only single interactions between light sources and surfaces
- First model the light sources in the scene
- Next build a reflection model that deals with interactions between materials and light

Local vs. Global Rendering

- Correct shading requires a global calculation involving all objects and light sources
- Incompatible with pipeline model which shades each polygon independently (local rendering)
- However, in computer graphics, especially in real-time graphics, we are happy if things "look right"
- There exist many techniques for approximating global effects

Light-Material Interaction

- Light that strikes an object is partially absorbed and partially scattered (reflected)
- The amount of reflected light determines the color and brightness of the object
- A surface appears red under white light because the red component of the light is reflected and the rest is absorbed
- The reflected light is scattered in a manner that depends on the smoothness and orientation of the surface

Light-Material Interaction

- Three types of surfaces are possible:
 - Specular Surfaces**: Appear shiny as most of reflected or scattered light is in a narrow range of angles close to angle of reflection
 - Translucent Surfaces**: Reflected light gets scattered in all directions
 - Diffuse Surfaces**: Allow some light to penetrate and gets refracted. Some light might also get reflected

Light Sources

- Light can leave a surface through two fundamental processes:
 - Self-emission
 - Reflection
- We will omit emissive term for our light model but it is easy to add self-emissive term
- General light sources are difficult to work with because we must integrate light coming from all points on the source

4.

UNIX Processes

UNIT 4: 15UC4C004 - UNIX System programming

main() function

- A C program starts execution with a function called `main()`
- When a C program is started by the kernel, a special start-up routine is called before `main()` is called
- Executable program file `file.o` specifies this start-up routine as the starting address for the program
- This is set up by the link editor when it is invoked by C compiler `gcc`
- The start-up routine takes values from kernel `__environ` and `__argv` and sets things up so that `main()` is called properly

Special Startup Routine

- The symbol `_start` is the entry point of C program
- The function with the name `_start` is supplied by a file called `entry.o` which contains the startup code for the C runtime environment
- Following actions are performed by the startup code:
 - Disable all interrupts
 - Copy any initialized data from ROM to RAM
 - Zero the uninitialized data area
 - Allocate space for and initialize the stack
 - Initialize the processor's stack pointer
 - Create and initialize the heap
 - Enable interrupts
 - Call `main()`

Process Termination

- There are five ways for a process to terminate:
 - Normal Termination
 - Return from `main()`
 - calling `exit()`
 - calling `_exit()`
 - Abnormal Termination
 - calling `abort()`
 - terminated by a signal
- Start-up routine is written so that if `main()` returns, `exit()` function is called

exit() and _exit() functions

- `exit()` returns the kernel immediately
- `_exit()` performs certain cleanup processing and then returns to the kernel
- Directly return to the kernel
- `void exit(int status);`
- `void _exit(int status);`
- `exit()` function has always performed a clean shutdown of the standard I/O library
- By calling `flush` on all open streams
- This causes all buffered output data to be flushed (written to file)

Environment List

- Each program is passed an environment list
- Environment list is an array of character pointers, with each pointer containing the address of a null-terminated C string
- Address of array of pointers is contained in global variable `environ`

exit() and _exit() functions

- Both functions expect a single integer argument called as exit status
- Most UNIX shells provide a way to examine the exit status of a process
- The exit status of a process is undefined if:
 - Either of `exit()` or `_exit()` is called without an exit status
 - `main()` does a return without return value
 - `main` falls off the end (an implicit return)

```

int main()
{
    printf("Hello world\n");
}
    
```

atexit() function

- With ANSI C, a process can register up to 32 functions that are automatically called by `exit()`
- These are referred to as exit handlers
- They are registered by calling `atexit()` function
- Returns 0 if OK, nonzero on error
- It takes address of a function as an argument
- `exit()` calls these functions in reverse order of their registration
- `exit()` first calls exit handlers and then `fclose()` all open streams

How C program is started and how it terminates

Environment List

- Program to print environment variables

```

#include <stdio.h>
int main()
{
    printf("Hello world\n");
}
    
```

Memory Layout of a C Program

- A C program comprises of the following pieces:
 - Text Segment**:
 - Contains machine instructions that are executed by the CPU
 - It is readable so that only a single copy needs to be in memory for frequently executed programs
 - It is read-only, so as to prevent a program from accidentally modifying its instructions
 - Uninitialized Data Segment**:
 - Also referred as `.bss` segment
 - Named after an ancient assembler that stood for `blank started by hand`
 - Data in this segment is initialized by the kernel to arithmetic 0 or null pointers before program starts execution
 - Stack**:
 - Contains automatic variables along with information that is saved each time a function is called
 - Each time a function is called, the address of where to return to and certain information about the caller's environment is saved on the stack

Memory Layout of a C Program

- Stack (cont.)**:
 - The newly called function then allocates room on the stack for its automatic and temporary variables
- Heap**:
 - Dynamic memory allocation takes place in this segment
 - Historically heap has been allocated between the top of `bss` segment and bottom of stack segment
- `size` command reports the size (in bytes) of text, data and `bss` segments

3) Sample recorded Video links

Sl. No	Name of the faculty	YouTube links
1.	Dr.S.M.Joshi	Computer Networks learning resources https://www.youtube.com/channel/UCkMvj_s4QYhplIE5kxwo1AQ
3.	Prof. Indira Umarji	Data Structure learning resources https://youtube.com/channel/UCtlW-cv1ZjVbX-ZBvQ_Gotg
4.	Prof. Nita G Kulkarni	ARM Processor learning resource https://www.youtube.com/watch?v=-7hncYmPr7Q
5.	Prof Sharada H N	ARM Processor learning resource https://bit.ly/SHNARM
6.	Prof. Ravindra Dastikop	Cloud Computing, IOT, Blockchain Learning and Teaching resources https://www.youtube.com/user/dastikop
7.	Prof. Govind Negalur	C Programming, LaTeX learning resources https://www.youtube.com/channel/UCsAGcViGvQoHWySbbQz8QDw



4) Sample Google Classroom Links:

Sl.No	Name of the faculty	Google Classroom links
1.	Dr. Vidyagouri. Kulkarni	https://classroom.google.com/c/NDA2MTMwODAyNzc1?cjc=c4552hg
2.	Prof. Indira Umarji	https://classroom.google.com/c/Mjk4NDczMzI5OTky?cjc=btusgt5
3.	Prof. Nita G Kulkarni	https://classroom.google.com/c/NDA2OTM4NzAzMTA3?cjc=ce6e3ii https://classroom.google.com/c/MzAwNDk3NTczODEy?cjc=rgxhp75
4.	Prof. Govind Negalur	https://classroom.google.com/c/NTQ1MzI3ODAyODY1 https://classroom.google.com/c/NDE1NTE1ODQwMDY5

5) Sample Blog Links:

Sl. No	Name of the Faculty	Blog link
1.	Prof. Ravindra Dastikop	Industrial orientation and programming and Practices: https://2018-iopp.blogspot.com Cloud Computing: https://cloudcomputingcourse.blogspot.com
2.	Prof. Govind Negalur	C Programming: https://18ucsc200-pspc.blogspot.com/ Advanced Java Programming: https://15ucsc601-aoop.blogspot.com/ Computer Graphics: https://cg-11ucsc601.blogspot.com/
3.	Prof. Pratap Kumar MK	Database Management Systems: https://dbms-15ucsc502.blogspot.com/ Web Technology: https://web2022-18ucso707.blogspot.com/

