Rajiv Gandhi University of Knowledge Technologies

LIVE LECTURE VISUAL GENERATION

Rajiv Gandhi University of Knowledge Technologies Srikakulam – 532402

Under the Guidance of

Mr. K DILEEP KUMAR

Asst. Professor, Dept of CSE

IIIT RGUKT, Srikakulam

Project done by:

S180443 – UDAY SANKAR GOTTIPALLI

S180681 – SANKARA RAO VANTAKU

S180020 – SAI KUMAR TAMMINANA

Live Lecture Visual Generation

A PROJECT REPORT

Submitted in partial fulfilment of requirements to

RGUKT - SRIKAKULAM

For the award of the degree

B. Tech in CSE

By

UDAY SANKAR GOTTIPALLI
(S180443),
SANKARA RAO VANATAKU
(S180681),
SAI KUMAR TAMMINANA
(S180020)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
RGUKT SRIKAKULAM, ETCHERLA
JULY 2023

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

DECLARATION

I certify that

a. The work contained in this report is original and has been done by us under the guidance of my supervisor(s).

b. The work has not been submitted to any other Institute for any degree or diploma.

c. We have followed the guidelines provided by the University in preparing the report.

d. We have confirmed the norms and guidelines given in the Ethical Code of Conduct of the University.

e. Whenever we have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references.

f. The work was done in the academic semester period i.e., from March 2023 – July 2023.

CERTIFICATE

Certified that this project work titled "**Live Lecture Visual Generation**" is the bona fide work of **Mr. Uday Sankar Gottipalli** bearing ID. No. **S180443**, **Mr. Sankara Rao Vantaku** bearing ID. No. **S180681** and **Mr. Sai Kumar Tamminana** bearing ID. No. **S180020** who carried out the work under my supervision and submitted in partial fulfilment of the requirements for the award of the degree, BACHELOR OF TECHNOLOGY, during the year 2022 – 2023.

Dileep Kuman Koda,
Project Guide,
Department of CSE,
RGUKT, SRIKAKULAM.

Sesha Kumar Nalluri,
Head of the Department,
Department of CSE,
RGUKT, SRIKAKULAM

Abstract:

In traditional lectures, students often struggle to keep up with the pace of the lecturer, leading to gaps in understanding and reduced engagement. Imagination is the crucial aspect of understanding a lecture. However, lack of imagination or knowledge to imagine or due to distraction while imagining the class - causes student to lose track of the concept. This project aims to develop a system that can automate the lecture into animation, creating an engaging and interactive learning experience. The objectives of this project are to develop a software that will recognise the lecturer voice, understand the lecture, convert the voice into an animated visual and evaluate the effectiveness of the system in improving student engagement and learning outcomes. There is some animation generation software existed like DreamCloud, Dall-e, Deep-AI. However, they can only convert some text into animated visual. This software is developing to recognise the voice and provide live animation during the lecture itself. This project will be the combination of technologies such as NLP, Machine learning and graphical tools such as Turtle and Canvas. NLP and Machine learning will be used to recognise and understand the lecture. The effectiveness of the system will be evaluated through user testing and analysis of student engagement and learning outcomes. The system is expected to significantly improve student engagement and learning outcomes compared to traditional lectures. The project will demonstrate the feasibility of using AI methodology and animation technology to automate lectures in ICT based education. The development of a system for automated lecture animation has the potential to revolutionize the way lectures are delivered and received. The project will provide important insights into the use of AI and animation technology in education, with implications for the development of new and innovative teaching methods.

Keywords: lecture automation, animation, AI, NLP, machine learning, student engagement, learning outcomes, Graphical Tools.

RAJIV GANDHI UNIVERSITY OF KNOWLEDGE TECHNOLOGIES

RGUKT-SRIKAKULAM, Srikakulam Dist. - 532402

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be incomplete without the mention of people whose perpetual cooperation made it viable, whose constant guidance and encouragement crown all efforts with success.

We would like to express our profound gratitude and deep regards to our guide **Mr. K Dileep Kumar** for his exemplary guidance, inspiration, monitoring, and constant encouragement with constructive suggestions throughout the course of this project.

We are extremely grateful for the confidence bestowed in us and for entrusting our project entitled “**Live Lecture Visual Generation**”. At this juncture, we feel deeply honoured in expressing our sincere thanks to her for providing valuable insights leading to the successful completion of our project.

S180443 – UDAY SANKAR GOTTIPALLI

S180681 – SANKARA RAO VANTAKU

S180020 – SAI KUMAR TAMMINANA

Table of Contents

CHAPTER 1

INTRODUCTION:

1.1. Purpose:

The purpose of this document is to provide a detailed description of the requirements for the development of a system that recognizes the lecturer's voice and generates effective visuals in real-time. The aims to address the challenges faced by students in traditional lectures by enhancing engagement and understanding through interactive visual representations.

1.2. Scope:

The scope of this project includes the development of a system that utilizes natural language processing (NLP) techniques and graphical tools like Canvas. The system will be capable of converting live voice into visuals, thereby providing students with a more interactive learning experience. The project will involve user testing, analysis of student engagement, and evaluation of learning outcomes to assess the effectiveness of the assistant.

CHAPTER 2

OVERALL DESCRIPTION:

2.1. Product Perspective:

The system will be a standalone system designed to integrate with existing lecture delivery platforms. It will receive live audio input, process it using NLP algorithms, and generate real-time visuals based on the lecture content.

2.2. Product Features

2.2.1. Voice Recognition:

The system will recognize the lecturer's voice to differentiate between speech and other audio sources.

2.2.2. Content Understanding:

The system will utilize NLP techniques to understand the lecture content and identify key concepts.

2.2.3. Real-time Visual Generation:

Based on the recognized voice and content understanding, the system will generate relevant visuals in real-time.

2.2.4. Interactive Visuals:

The generated visuals will be interactive, allowing students to explore and engage with the lecture material.

2.2.5. Integration:

The assistant will integrate seamlessly with existing lecture delivery platforms to facilitate easy adoption and usage.

2.3. User Classes and Characteristics

2.3.1. Lecturers:

They will use the system as a tool to enhance their lectures by providing visual aids in real-time.

2.3.2. Students:

They will benefit from the interactive visuals generated by the system to improve engagement and understanding.

2.4. Operating Environment:

The GUI can be accessed in any system with Python 3.6 version or above

The required modules are:

1. Spacy
2. Speech_recognition
3. Shapely
4. Tkinter
5. Math
6. Randon
7. String
8. nltk

The developers operating environment are:

1. Tkinter for testing.
2. Tkinter for displaying output of the Lecture

2.5. Technologies Used

1. Python
2. Natural Language Processing
3. Graphical Tools - Canvas

2.6. Requirements

1. Software Requirements:
    a. Python 3.6
    b. Libraries such as speech_recognition, spacy, nltk, shapely
    c. Windows 8/9/10 or Linux OS
2. Hardware Requirements:
    a. Laptop/Desktop
    b. 4GB RAM

CHAPTER 3

FEASIBILITY STUDY:

3.1. Requirements:

A key part of the preliminary investigation is that the system can be used to increase efficiency in the different aspects regarding the ICT based education.

3.2. Economic Feasibility:

The system does not require any special hardware, only the computer is needed with specified requirement specifications.

3.3. Behavioural Feasibility:

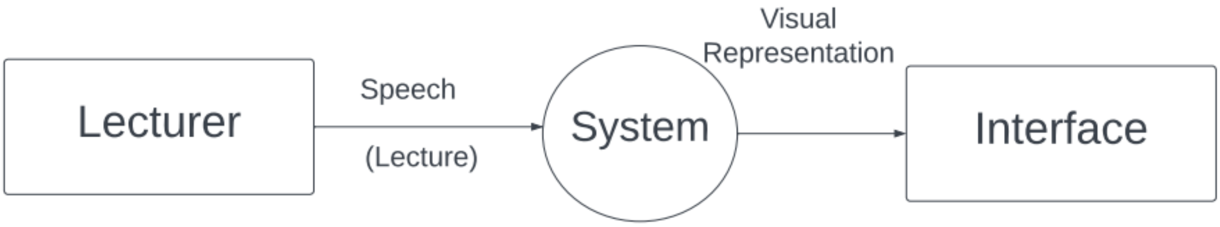
The system produces mostly relevant visuals to the class held by lecture in real-time. However, the system is quite sensitive to ambiguous context and mispronunciations. They could lead to wrong generations of the visual and they could sometimes mislead the classes.

CHAPTER 4

DESIGN:

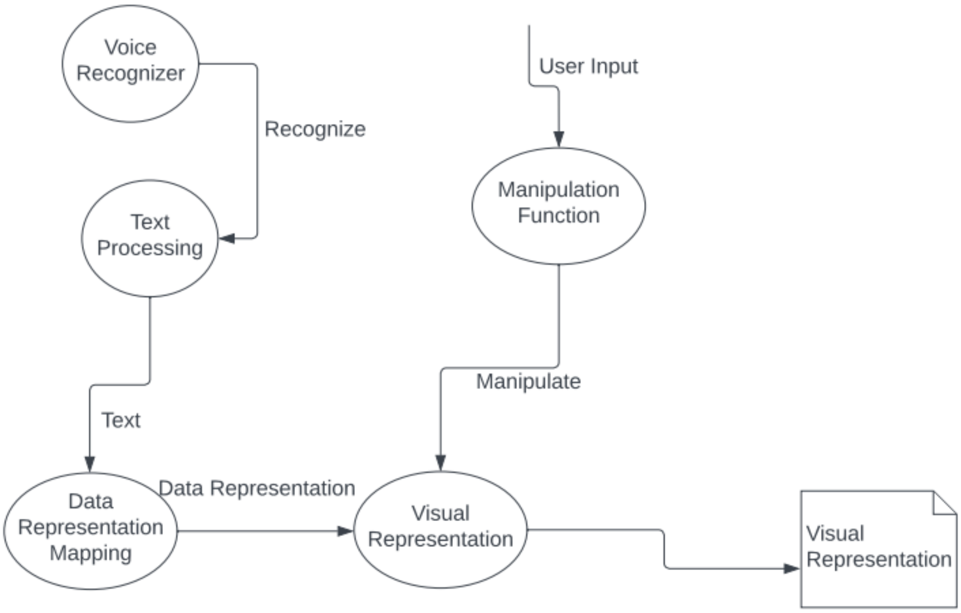
4.1. Data Flow Diagrams

DFD Level - 0

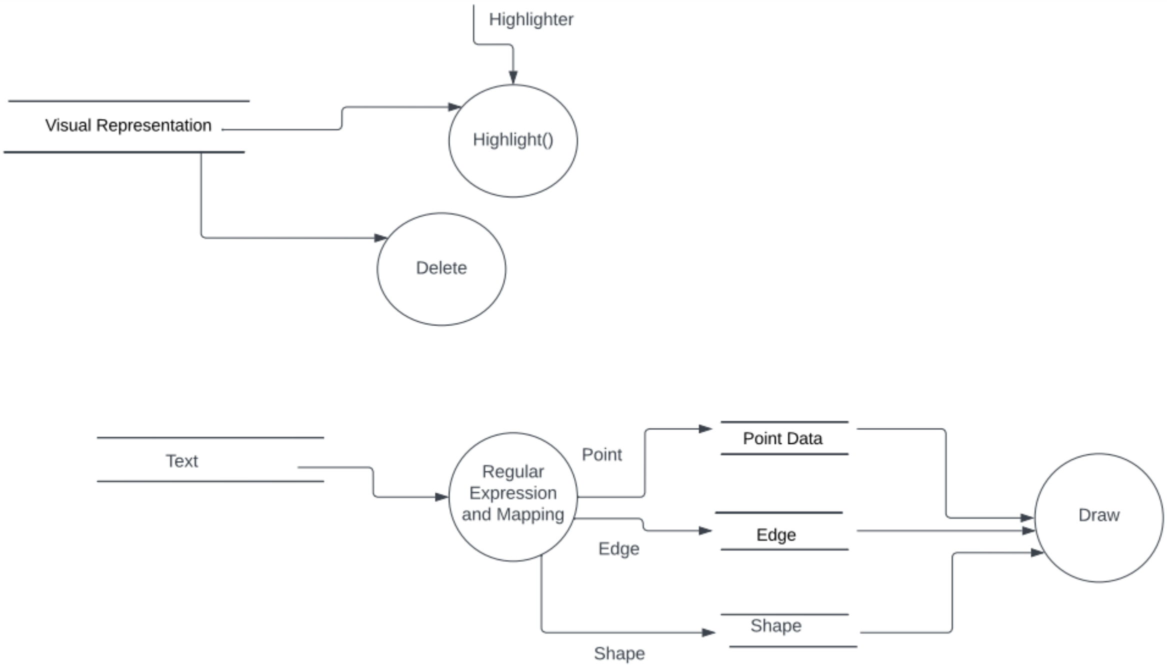


Level- 0

Level 1

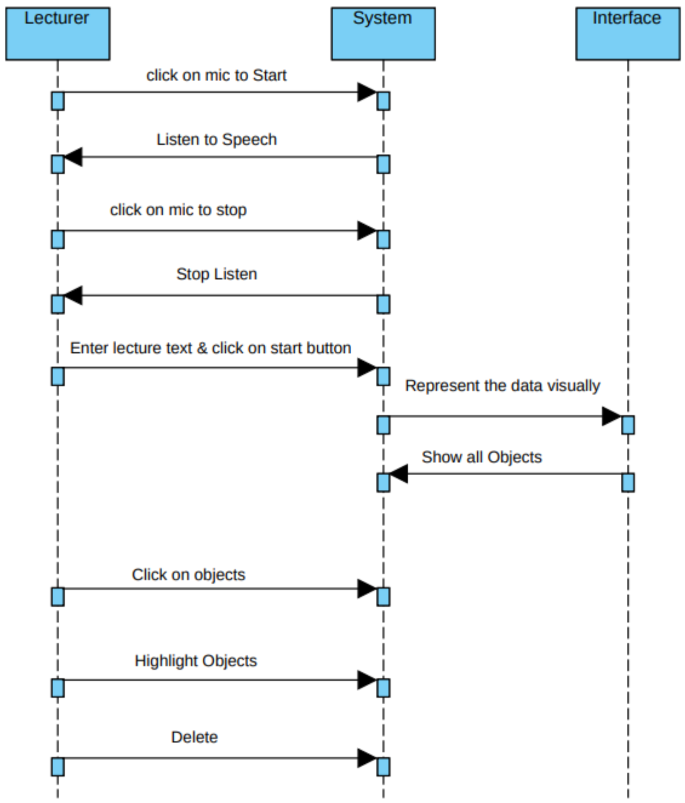


Level 2

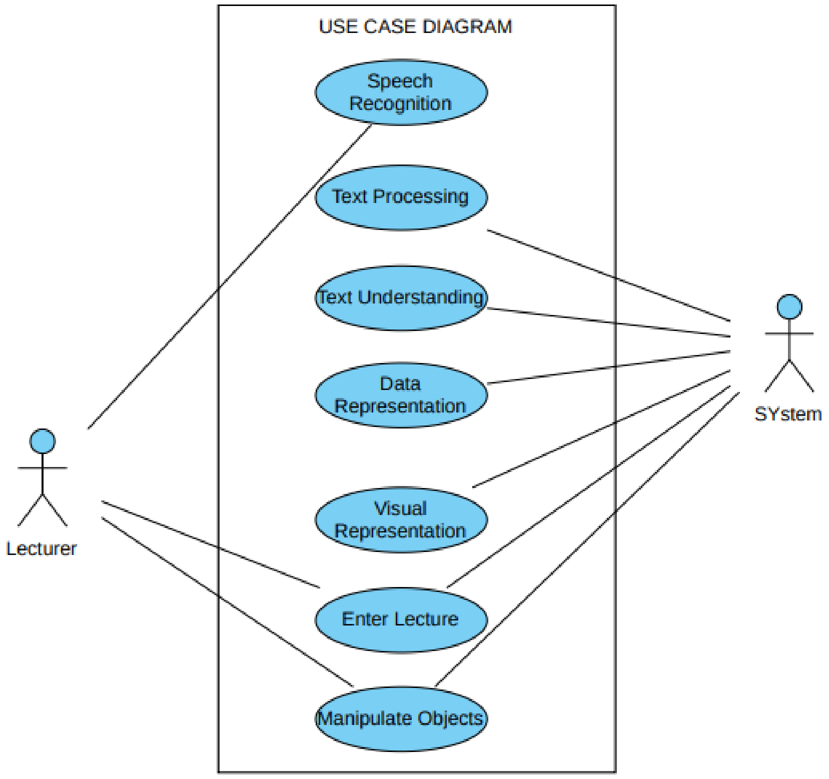


4.2. UML Diagrams

4.2.1. Sequence Diagram



4.2.2. Use case Diagram



CHAPTER 5

IMPLEMENTATION

Steps to Build the Model

1. Create the Data Representation classes (Point, Edge, Angle, Shape)
2. Speech recognition of the lecturer
3. Text processing
4. Mapping the text to the data representation
5. Interface creation
6. Visual generation

5.1. Create the Data Representation classes (Point, Edge, Angle, Shape)

Point.py:

```
import random

cur_points = {}
temp_point_n = 0


class Point:
  def __init__(self, x=None, y=None, z=None, tags=None,
         visibility=1, highlighting=False, fillcolor='black', radius=4):
    self.highlighting = highlighting
    self.fillcolor = fillcolor
    self.radius = radius
    self.canvas_id = None
    if x is not None:
      self.x = x
    else:
      self.x = random.randint(-500, 500)
    if y is not None:
      self.y = y
    else:
      self.y = random.randint(-300, 300)
    self.z = z
```

```
        if tags is not None:
            self.tags = tags
        else:
            global temp_point_n
            self.tags = 't_p_' + str(temp_point_n)
            temp_point_n += 1

        self.visibility = visibility
        cur_points[self.tags] = self

    def draw(self, canvas):
        if self.visibility == 1:
            self.canvas_id = canvas.create_oval(self.x - self.radius, self.y - self.radius, self.x + self.radius,
                                                self.y + self.radius, fill=self.fillcolor, tags=self.tags)


# change the coordinates of the points
def changeCoords(tag, x=None, y=None):
    for tags in cur_points.keys():
        for t in tags.split():
            if t == tag:
                if x is not None:
                    cur_points[tags].x = x
                if y is not None:
                    cur_points[tags].y = y



def changeTurtleCoords(x=None, y=None):
    cur_points['turtle'].x = x
    cur_points['turtle'].x = y



def movePoint(cvs, tag, dx, dy):
    objects = cvs.find_withtag(tag)
    for obj in objects:
        # Move the point by dx and dy
        cvs.move(obj, dx, dy)
    cvs.after(50, movePoint, cvs, tag, dx, dy)  # Repeat the animation after a delay



def increment_temp_point_n(s):
    global temp_point_n
    temp_point_n += 1
    s.temp_point_n += 1
```

Edge.py

```
from functoins import *

cur_edges = {}
temp_edge_n = 0



class Edge:
    def __init__(self, p1=None, p2=None, length=None, slope=None, tag=None, fill_color='black', visibility=1,
                width=1, highlighting=False):
```

```
self.highlighting = highlighting
self.width = width
self.canvas_id = None
if tag is not None:
    self.tags = tag
else:
    global temp_edge_n
    self.tags = 't_ed' + str(temp_edge_n)
    temp_edge_n += 1
if fill_color is not None:
    self.fill_color = fill_color
else:
    self.fill_color = 'black'
if p1 is not None:
    self.p1 = cur_points[p1]
    if p2 is not None:
        self.p2 = cur_points[p2]
        self.length = math.sqrt((self.p1.x - self.p2.x) ** 2 + (self.p1.y - self.p2.y) ** 2)
        if self.p2.x == self.p1.x:
            self.slope = float('inf')
        else:
            self.slope = (self.p2.y - self.p1.y) / (self.p2.x - self.p1.x)
    else:
        if length is not None:
            self.length = length
        else:
            self.length = 200
        if slope is not None:
            self.slope = slope
        else:
            self.slope = 0
        temp_x, temp_y = second_point(self.p1.x, self.p1.y, self.length, self.slope)
        self.p2 = Point(x=temp_x, y=temp_y)
else:
    if p2 is not None:
        self.p2 = cur_points[p2]
        if length is not None:
            self.length = length
        else:
            self.length = 200
        if slope is not None:
            self.slope = slope
        else:
            self.slope = 0
        self.p1 = Point(second_point(self.p2.x, self.p2.y, self.length, self.slope))
    else:
        if length is not None:
            self.length = length
        else:
            self.length = 200
        if slope is not None:
            self.slope = slope
        else:
            self.slope = 0
        self.p1 = Point(x=randonNum(-100, 100), y=randonNum(-100, 100))
        p2 = second_point(self.p1.x, self.p1.y, self.length, self.slope)
        self.p2 = Point(p2[0], p2[1])

self.visibility = visibility
cur_edges[self.tags] = self
```

```
    def draw(self, canvas):
        if self.visibility == 1:
            self.canvas_id = canvas.create_line(self.p1.x, self.p1.y, self.p2.x, self.p2.y, fill=self.fill_color,
                                                width=self.width, tags=self.tags)
```

Angle.py

```
cur_angles = {}


class Angle:
    def __init__(self, angle=None, ini_point=None, vertex=None, end_point=None,
                 start_angle=None, end_angle=None, tag=None, visibility=1, highlighting=False):
        self.highlighting = highlighting
        self.angle = angle
        self.ini_point = ini_point
        self.vertex = vertex
        self.end_point = end_point
        self.start_angle = start_angle
        self.end_angle = end_angle
        self.tags = tag
        self.visibility = visibility
        self.canvas_id = None
        if self.vertex is None or self.ini_point is None or self.end_point is None:
            self.visibility = 0
        cur_angles[self.tags] = self
```

Shape.py

```
from Edge import *
from Angle import *

cur_polygons = {}
temp_shape_n = 0

class Poly:
    def __init__(self, count=3, edges=None, points=None, angles_tags=None, angles=None, base_length=None,
                 base_edge=None, base_angle=None, tag=None, visibility=1, highlighting=False):
        self.highlighting = highlighting
        self.edges = []
        self.angles = []
        self.points = []
        self.visibility = visibility
        self.count = count
        self.base_edge = None
        self.canvas_id = None

        if tag is not None:
            self.tags = tag
        else:
            global temp_shape_n
            self.tags = 't_sh_' + str(temp_shape_n)
            temp_shape_n += 1
```

```
if count is not None:
    self.count = count
    if edges is not None and len(edges) == 1 and base_edge is None:
        self.base_edge = edges[0]
        if base_length is None:
            base_length = cur_edges[self.base_edge].length
            self.base_length = base_length
    if points is not None and len(points) >= self.count:
        self.points = points[:self.count]
        points_order(self)
        append_edges(self)
        append_angles(self)

    elif edges is not None and len(edges) >= self.count:
        self.edges = edges[:self.count]
        for i in range(self.count):
            self.points.append(cur_edges[self.edges[i]].p1.tags)
        points_order(self)
        append_angles(self)

    elif angles_tags is not None and len(angles_tags) >= self.count:
        self.angles_tags = angles_tags[self.count]
        if self.base_edge is None:
            if base_length is not None:
                self.base_length = base_length
            else:
                self.base_length = 150
            create_baseedge(self)
        else:
            self.points.append(cur_edges[self.base_edge].p1)
            self.points.append(cur_edges[self.base_edge].p2)
        find_points(self, with_tag=True)
        append_edges(self)
        append_angles(self)

    else:
        if angles is not None and len(angles) == self.count:
            self.angles = angles
        else:
            if base_angle is not None:
                self.base_angle = base_angle
            else:
                self.base_angle = findRegularPolygonAngle(self.count)
            self.angles = [self.base_angle] * self.count
        if self.base_edge is None:
            if base_length is not None:
                self.base_length = base_length
            else:
                self.base_length = 150
            create_baseedge(self)
        else:
            self.points.append(cur_edges[self.base_edge].p1.tags)
            self.points.append(cur_edges[self.base_edge].p2.tags)
        find_points(self, with_tag=False)
        append_edges(self)
        append_angles(self)

cur_polygons[self.tags] = self
```

```
def points_order(s):
    if not is_polygon(s.points):
        s.points = reorder_points(s.points)
        s.points = [i.tags for i in s.points]
        if polygon_orientation(s.points) == 'counterclockwise':
            s.points = s.points[::-1]


def append_edges(s):
    for i in range(s.count):
        vertex = s.points[i % s.count]
        end_point = s.points[(i + 1) % s.count]

        temp_edge = Edge(p1=vertex,
                         p2=end_point,
                         tag=vertex + end_point)
        s.edges.append(temp_edge.tags)


def append_angles(s):
    s.angles = []
    for i in range(s.count):
        ini_point = s.points[(i - 1) % s.count]
        vertex = s.points[i % s.count]
        end_point = s.points[(i + 1) % s.count]

        angle_tag = ini_point + vertex + end_point
        s.angles.append(angle_tag)

        start_angle = edge_inclination(vertex, ini_point)
        end_angle = edge_inclination(vertex, end_point)

        if start_angle < end_angle:
            extent = end_angle - start_angle
        elif start_angle > end_angle:
            extent = end_angle + 360 - start_angle
        else:
            extent = 0

        Angle(angle=extent, ini_point=ini_point, vertex=vertex, end_point=end_point, start_angle=start_angle,
              end_angle=end_angle, tag=angle_tag)


def find_points(s, with_tag):
    con_angles = []
    static_angle = 0
    for i in range(s.count):
        if with_tag:
            static_angle += (180 - cur_angles[s.angles_tags[i]].angle)
        else:
            static_angle += (180 - s.angles[i])
        con_angles.append(static_angle)
    for i in range(s.count - 2):
        t1, t2 = find_point_c(cur_points[s.points[i + 1]].x, cur_points[s.points[i + 1]].y, cur_points[s.points[i]].x,
                              cur_points[s.points[i]].y, con_angles[i])
        t_p_1 = Point(t1, t2)
        s.points.append(t_p_1.tags)


def create_baseedge(s):
    t_p_1 = Point(x=randonNum(-100, 100), y=randonNum(-100, 100))
    s.points.append(t_p_1.tags)
    t_p_2 = Point(x=(t_p_1.x + s.base_length), y=t_p_1.y)
    s.points.append(t_p_2.tags)


def findRegularPolygonAngle(n):
    if n < 3:
        print("A polygon must have at least 3 sides.")
    else:
        return (n - 2) * 180 / n
```

functions.py

```
import math
from shapely.geometry import Polygon

from Point import *


def randonNum(low, high):
    return random.randint(low, high)



def second_point(x, y, lg, s):
    # calculate change in x and y
    if s == float('inf'):
        delta_x = 0
        delta_y = lg
    else:
        delta_x = math.sqrt(lg ** 2 / (1 + s ** 2))
        delta_y = s * delta_x

    # calculate x2 and y2
    x2_1 = x + delta_x
    y2_1 = y + delta_y

    return x2_1, y2_1



def find_point_c(x1, y1, x2, y2, theta):
    # calculate distance AB
    ab = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
    # calculate vector AC with magnitude BC and angle theta
    bc = ab  # assuming BC = AB
    v = [math.cos(math.radians(theta)), math.sin(math.radians(theta))]
    ac = [bc * v[i] for i in range(2)]

    # calculate coordinates of point C
    x3 = x1 + ac[0]
    y3 = y1 + ac[1]
    return x3, y3


def calculate_orientation(p, q, r):
    # Calculate the polygon_orientation of three points (p, q, r)
    val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y)
    if val == 0:
        return 0
    elif val > 0:
        return 1
    else:
        return -1


def reorder_points(points):
    points = [cur_points[i] for i in points]
    n = len(points)

    # Check if there are at least 3 points
    if n < 3:
        return []

    # Find the leftmost point (point with minimum x-coordinate)
    leftmost = min(points, key=lambda p: p.x)

    # Sort the remaining points based on their polar angle in counterclockwise order
    sorted_points = sorted(points, key=lambda p: (
        calculate_orientation(leftmost, p, Point(leftmost.x, leftmost.y + 1)),
        -((p.y - leftmost.y) / (p.x - leftmost.x) if p.x != leftmost.x else float('inf'))
    ))

    return sorted_points


def polygon_orientation(points):
    # Calculate the cross product of consecutive edges
    polygon = [[cur_points[p].x, cur_points[p].y] for p in points]
    cross_product_sum = 0
    for i in range(len(polygon)):
        current_point = polygon[i]
        next_point = polygon[(i + 1) % len(polygon)]
        cross_product_sum += (next_point[0] - current_point[0]) * (next_point[1] + current_point[1])

    # Determine the polygon_orientation based on the cross product sum
    if cross_product_sum > 0:
        return "counterclockwise"
    elif cross_product_sum < 0:
        return "clockwise"
    else:
        return "collinear"


def is_polygon(points):
    points = [cur_points[i] for i in points]
    polygon = Polygon([[p.x, p.y] for p in points])
    return polygon.is_valid
```

```
def calculate_angle(a, b, c):
    a = cur_points[a]
    b = cur_points[b]
    c = cur_points[c]
    # Calculate vectors AB and BC
    vector_ab = [a.x - b.x, a.y - b.y]
    vector_bc = [c.x - b.x, c.y - b.y]

    dot_product = vector_ab[0] * vector_bc[0] + vector_ab[1] * vector_bc[1]

    magnitude_ab = math.sqrt(vector_ab[0] ** 2 + vector_ab[1] ** 2)
    magnitude_bc = math.sqrt(vector_bc[0] ** 2 + vector_bc[1] ** 2)

    angle_rad = math.acos(dot_product / (magnitude_ab * magnitude_bc))
    angle_deg = math.degrees(angle_rad)

    return angle_deg


def edge_inclination(sp, ep):
    start_point = cur_points[sp]
    end_point = cur_points[ep]
    dx = end_point.x - start_point.x
    dy = end_point.y - start_point.y
    if dx == 0:
        if start_point.y > end_point.y:
            angle_degrees = 90
        elif start_point.y < end_point.y:
            angle_degrees = 270
        else:
            angle_degrees = 0
    elif dy == 0:
        if start_point.x < end_point.x:
            angle_degrees = 0
        elif start_point.x > end_point.x:
            angle_degrees = 180
        else:
            angle_degrees = 0
    else:
        angle = math.atan(dy / dx)  # Calculate the angle in radians
        angle_degrees = math.degrees(angle)  # Convert the angle to degrees
        if start_point.x < end_point.x:
            if start_point.y > end_point.y:
                angle_degrees *= -1
            elif start_point.y < end_point.y:
                angle_degrees = 360 - angle_degrees
        elif start_point.x > end_point.x:
            if start_point.y > end_point.y:
                angle_degrees = 180 - angle_degrees
            elif start_point.y < end_point.y:
                angle_degrees *= -1
                angle_degrees = 180 + (angle_degrees % 370)

    return angle_degrees


def point_in_direction(v, sp, ep, n):
    vertex = cur_points[v]
    start_point = cur_points[sp]
    end_point = cur_points[ep]

    direction_vector = (end_point.x - start_point.x, end_point.y - start_point.y)

    # Step 2: Normalize direction vector
    magnitude = math.sqrt((end_point.x - start_point.x) ** 2 + (end_point.y - start_point.y) ** 2)
    normalized_vector = (direction_vector[0] / magnitude, direction_vector[1] / magnitude)

    # Step 3: Calculate displacement vector
    displacement_vector = (normalized_vector[0] * n, normalized_vector[1] * n)

    # Step 4: Calculate point coordinates
    point_x = vertex.x + displacement_vector[0]
    point_y = vertex.y + displacement_vector[1]

    return point_x, point_y
```

5.2. Speech Recognition of the Lecturer

classLecture.py

```
import speech_recognition as sr

lecture = []
recognized = ''
listening = False
recognizer = None
stop_listening = None


def callback(recognizer, source):

    try:
        global recognized
        recognized = recognizer.recognize_google(source)
        global lecture
        lecture.append(recognized)
        print("you said ", lecture)

    except sr.RequestError as exc:
        print(exc)

    except sr.UnknownValueError:
        print("unable to recognize")
```

```
def toggle_listening(txt_field):
    global listening
    global recognizer
    global stop_listening

    if listening:
        txt_field.config(state='normal')
        stop_listening()  # Stop the listening loop
        listening = False
        print("Microphone listening stopped")
    else:
        txt_field.config(state='disabled')
        recognizer = sr.Recognizer()
        mic = sr.Microphone()
        with mic:
            recognizer.adjust_for_ambient_noise(mic, duration=0.1)
        print('Talk')
        stop_listening = recognizer.listen_in_background(mic, callback)
        listening = True
        print("Microphone listening started")


def reset_lecture():
    global lecture
    lecture = []
```

5.3. Text processing

Text –processing.py

```
import spacy
import string

nlp = spacy.load('en_core_web_sm')


def remove_punctuation(sentence):
    translator = str.maketrans("", "", string.punctuation)
    return sentence.translate(translator)


def Numbers(doc):
    numbers = []
    i = 0
    while i < len(doc):
        if doc[i].like_num or doc[i].ent_type_ == 'CARDINAL':  # Check if the token is a numerical value
            number = doc[i].text
            j = i + 1
            while j < len(doc):
                if doc[j].like_num or doc[j].ent_type_ == 'CARDINAL':
                    number = number + " " + doc[j].text
                    j += 1
                else:
                    break
            i = j + 1
            numbers.append(number)
        else:
            i += 1

    return numbers


def Verbs(doc):
    return [token.lemma_ for token in doc if token.pos_ == "VERB"]


def Nouns(doc):
    return [token.text for token in doc if token.pos_ == "NOUN"]
```

5.4. Mapping the text to the data representation

5.5. Interface Creation

5.6. Visual Generation (designed in single python doc)

Main.py

```
# importing necessary modules


import re

from PIL import Image, ImageTk

import tkinter as tk

import nltk


from text_processing import *

from classLecture import *

from Shape import *

from functoins import *

from animation import *


nlp = spacy.load('en_core_web_sm')


# variables


object_btn = {}
```

```
points_btn_flag = 0

edges_btn_flag = 0

sh_btn_flag = 0

popup_opened = False

popup = None

processed_lecture = []


action = None

create = ['create', 'consider', 'assume', 'imagine', 'draw', 'show', 'locate', 'suppose', 'picture', 'let']

classes = ['point', 'edge', 'angle', 'shape', 'polygon']


size = ['length', 'size', 'width', 'centimetres', 'millimetres', 'kilometres', 'metres', 'm', 'cm', 'mm', 'km']

inclination = ['slope', 'gradient', 'incline', 'inclination', 'pitch', 'grade', 'slant', 'tilt', 'steepness',
'descent']


each = ['each', 'any', 'every', 'all']

side = ['side', 'length', 'edge', 'line', 'boundary']

angle = ['angle', 'degree']


shapes = ['polygon', 'rectangle', 'square', 'triangle', 'pentagon', 'quadrilateral', 'circle', 'shape']

ed_3_sh = ['triangle']

ed_4_sh = ['quadrilateral', 'square', 'rectangle', 'rhombus', 'parallelogram', 'trapezoid', 'trapezium',
'kite',

        'isosceles trapezoid', 'irregular quadrilateral']

ed_5_sh = ['pentagon', 'regular pentagon', 'irregular pentagon', 'convex pentagon', 'concave
pentagon']

ed_6_sh = ['hexagon', 'regular hexagon', 'irregular hexagon', 'convex hexagon', 'concave hexagon',

        'regular hexagonal prism']

ed_7_sh = ['heptagon', 'regular heptagon', 'irregular heptagon', 'convex heptagon', 'concave
heptagon']

ed_8_sh = ['octagon', 'regular octagon', 'irregular octagon', 'convex octagon', 'concave octagon',

        'regular octagonal prism']

ed_9_sh = ['nonagon', 'regular nonagon', 'irregular nonagon', 'convex nonagon', 'concave nonagon']

ed_10_sh = ['decagon', 'regular decagon', 'irregular decagon', 'convex decagon', 'concave decagon']
```

```
cur_sentence = 0

# interface using tkinter

window = tk.Tk()


# full screen of interface

def ExitFullScreen(event):
    window.attributes("-fullscreen", False)


# Set the window size to full screen
window.attributes("-fullscreen", True)

# Bind the Escape key to exit full screen
window.bind("<Escape>", ExitFullScreen)


# important functions


def reset():
    global processed_lecture, cur_sentence, object_btn, points_btn_flag
    global edges_btn_flag, sh_btn_flag, popup_opened, popup

    cur_points.clear()
    cur_polygons.clear()
    cur_edges.clear()
    cur_angles.clear()
    processed_lecture = []
    object_btn = {}
    points_btn_flag = 0
    edges_btn_flag = 0
    sh_btn_flag = 0
    popup_opened = False
    popup = None
    processed_lecture = []


def append_lecture():
    txt = input_field.get("1.0", "end-1c")
    sentences = nltk.sent_tokenize(txt)
    lecture.extend(sentences)
    empty_textfield()


def empty_textfield():
    input_field.delete("1.0", tk.END)


# Remove the item from the canvas

def remove_item(item_id):
    if item_id in cur_points.keys():
        canvas.delete(cur_points[item_id].canvas_id)
    elif item_id in cur_edges.keys():
        canvas.delete(cur_edges[item_id].canvas_id)
    elif item_id in cur_polygons.keys():
        for point in cur_polygons[item_id].points:
            remove_item(point)
        for edge in cur_polygons[item_id].edges:
            remove_item(edge)
    elif item_id in cur_angles.keys():
        canvas.delete(cur_angles[item_id].canvas_id)


def highlight_point(point):
    if cur_points[point].highlighting:
        cur_points[point].fillcolor = '#FAFF9A'
        cur_points[point].radius = 6
    else:
        cur_points[point].fillcolor = 'black'
        cur_points[point].radius = 4

    remove_item(point)
    cur_points[point].canvas_id = None
    showPoint(point)


def highlight_edge(item):
    if cur_edges[item].highlighting:
        cur_edges[item].fill_color = '#FFD500'
        cur_edges[item].width = 3
    else:
        cur_edges[item].fill_color = 'black'
        cur_edges[item].width = 1

    remove_item(item)
    cur_edges[item].canvas_id = None
    showEdge(item)


def highlight_shape(item):
    if cur_polygons[item].highlighting:
        for edge in cur_polygons[item].edges:
            cur_edges[edge].highlighting = True
            highlight_edge(edge)
        for point in cur_polygons[item].points:
            cur_points[point].highlighting = True
            highlight_point(point)
    else:
        for edge in cur_polygons[item].edges:
            cur_edges[edge].highlighting = False
            highlight_edge(edge)
        for point in cur_polygons[item].points:
            cur_points[point].highlighting = False
            highlight_point(point)


def highlight_item(item):
    if item in cur_points.keys():
        cur_points[item].highlighting = not cur_points[item].highlighting
        highlight_point(item)
    elif item in cur_edges.keys():
        cur_edges[item].highlighting = not cur_edges[item].highlighting
        highlight_edge(item)
    elif item in cur_polygons.keys():
        cur_polygons[item].highlighting = not cur_polygons[item].highlighting
        highlight_shape(item)


def create_obj_btn(name, ele):
    object_btn[name] = tk.Button(ele, text=name, height=1, width=35,
                                 bg='#0066CC', fg='#B8965E')
    object_btn[name].bind("<Button-1>", lambda event: open_popup(event, name))
    object_btn[name].pack()
    object_btn[name].config(font=("Arial", 12, "bold"))



def destroy_btn(name):
    if name in object_btn.keys():
        object_btn[name].destroy()



def open_popup(event, name):
    global popup, popup_opened

    if popup_opened:
        popup.destroy()
        popup_opened = False
    else:
        popup = tk.Toplevel(window)
        popup.geometry("+{}+{}".format(event.x_root - (window.winfo_x() +
object_btn[name].winfo_x()) - 220,
                                   event.y_root))
        popup.overrideredirect(True)  # Removes the window decorations

        # Create the buttons in the popup
        highlight = tk.Button(popup, height=1, width=20, text="highlight",
                              command=lambda: highlight_item(name), bg='#0066CC', fg='#FAFF9A')
        highlight.pack()
        highlight.config(font=("Arial", 12, "bold"))

        delete = tk.Button(popup, height=1, width=20, text="delete",
                           command=lambda: remove_item(name), bg='#0066CC', fg='#FAFF9A')
        delete.pack()
        delete.config(font=("Arial", 12, "bold"))
        popup_opened = True



def des_all_obj_btns():
    global points_btn_flag, edges_btn_flag, sh_btn_flag
    for i in cur_points.keys():
        destroy_btn(i)
    points_btn_flag = 0
    for i in cur_edges.keys():
        destroy_btn(i)
    edges_btn_flag = 0
    for i in cur_polygons.keys():
        destroy_btn(i)
    sh_btn_flag = 0



def show_objects(mode):
    global points_btn_flag, edges_btn_flag, sh_btn_flag

    if mode == 'points':
        if points_btn_flag == 0:
            for i in cur_points.keys():
                create_obj_btn(i, point_objects_div)
            points_btn_flag = 1
else:
            for i in cur_points.keys():
                object_btn[i].destroy()
            points_btn_flag = 0
    if mode == 'edges':
        if edges_btn_flag == 0:
            for i in cur_edges.keys():
                create_obj_btn(i, edge_objects_div)
            edges_btn_flag = 1
        else:
            for i in cur_edges.keys():
                destroy_btn(i)
            edges_btn_flag = 0
    if mode == 'shapes':
        if sh_btn_flag == 0:
            for i in cur_polygons.keys():
                create_obj_btn(i, sh_objects_div)
            sh_btn_flag = 1
        else:
            for i in cur_polygons.keys():
                destroy_btn(i)
            sh_btn_flag = 0



def animateCanvas():
    global cur_sentence
    if cur_sentence < len(lecture):
        if lecture[cur_sentence] not in processed_lecture:
            prompt(lecture[cur_sentence])
            processed_lecture.append(lecture[cur_sentence])
        cur_sentence += 1
    canvas.after(10, animateCanvas)



def showPoint(tag):
    for t in cur_points.keys():
        if t == tag:
            obj = cur_points[t]
            if obj.visibility == 1:
                obj.canvas_id = canvas.create_oval(obj.x - obj.radius, obj.y - obj.radius, obj.x + obj.radius,
                                                   obj.y + obj.radius, fill=obj.fillcolor, tags=obj.tags)


def showEdge(tag):
    for t in cur_edges.keys():
        if t == tag:
            obj = cur_edges[t]
            if obj.visibility == 1:
                if obj.tags == 'x-axis' or obj.tags == 'y-axis':
                    obj.canvas_id = canvas.create_line(obj.p1.x, obj.p1.y, obj.p2.x, obj.p2.y, fill=obj.fill_color,
                                                       width=obj.width)
                else:
                    draw_line(canvas, obj)


def showAngle(tag):
    for t in cur_angles.keys():
        if t == tag:
            obj = cur_angles[t]
            if obj.visibility == 1:
                showEdge(obj.ini_point + obj.vertex)
                showEdge(obj.vertex + obj.end_point)
                n = 20
                obj.canvas_id = canvas.create_arc(cur_points[obj.vertex].x - math.sqrt(2) * n,
                                                  cur_points[obj.vertex].y - math.sqrt(2) * n,
                                                  cur_points[obj.vertex].x + math.sqrt(2) * n,
                                                  cur_points[obj.vertex].y + math.sqrt(2) * n,
                                                  start=obj.start_angle, extent=obj.angle, style=tk.ARC)


def showPolygon(tag):
    for t in cur_polygons.keys():
            if t == tag:
                obj = cur_polygons[t]
                if obj.visibility == 1:
                    for i in obj.points:
                        showPoint(i)
                    for i in obj.edges:
                        showEdge(i)



def remove(tag):
    for tags in cur_points.keys():
        for t in tags.split():
            if t == tag:
                obj = cur_points[tags]
                obj.visibility = 0



def drawAllPoints():
    for point in cur_points.values():
        showPoint(point.tags)



def drawAllEdges():
    for edge in cur_edges.values():
        showEdge(edge.tags)



def drawAllAngles():
    for ang in cur_angles.values():
        showAngle(ang.tags)

def createPoint(x=None, y=None, z=None, tag=None):
    Point(x=x, y=y, z=z, tags=tag)



def create_point(txt, doc):
    tags = re.findall(r'[A-Z]*', txt)
    tags = list(filter(lambda x: x != '', tags))
    tags = sorted(tags, key=len)
    if len(tags) == 0:
        tags.append(None)
    coords = Numbers(doc)
    if len(coords) == 2:
        if re.search(r'\b[xX]\b.*\b[yY]\b', txt):
            createPoint(x=float(coords[0]), y=float(coords[1]), tag=tags[0])
        elif re.search(r'\b[yY]\b.*\b[xX]\b', txt):
            createPoint(x=float(coords[1]), y=float(coords[0]), tag=tags[0])
        else:
            createPoint(x=float(coords[0]), y=float(coords[1]), tag=tags[0])
    elif len(coords) == 1:
        only_x = r'\bx\b.*(\by\b).*(unknown|not given|not specified)'
        only_y = r'\by\b.*(\bx\b).*(unknown|not given|not specified)'
        not_x = r'\bx\b.*(unknown|not given|not specified).*\by\b'
        not_y = r'\by\b.*(unknown|not given|not specified).*\bx\b'
        for_x = r'\b[xX]\b'
        for_y = r'\b[yY]\b'
        if re.search(only_x, txt):
            createPoint(x=float(coords[0]), tag=tags[0])
        elif re.search(only_y, txt):
            createPoint(y=float(coords[0]), tag=tags[0])
        elif re.search(not_x, txt):
            createPoint(y=float(coords[0]), tag=tags[0])
        elif re.search(not_y, txt):
            createPoint(x=float(coords[0]), tag=tags[0])
        elif re.search(for_x, txt):
            createPoint(x=float(coords[0]), tag=tags[0])
        elif re.search(for_y, txt):
            createPoint(y=float(coords[0]), tag=tags[0])
    elif re.search(r'\bx[-_\s]axis\b', txt):
        createPoint(y=0, tag=tags[0])
    elif re.search(r'\by[-_\s]axis\b', txt):
        createPoint(x=0, tag=tags[0])
    elif re.search(r'\borigin\b', txt):
        createPoint(x=0, y=0, tag=tags[0])


def create_edge(txt, doc):
    p1 = None
    p2 = None
    tag = None
    fill_color = 'red'
    length = None
    slope = None
    visibility = 1
    tags = re.findall(r'[A-Z]*', txt)
    tags = list(filter(lambda x: x != '', tags))
    tags = sorted(tags, key=len)

    size_pattern = '(' + '|'.join(size) + ')'
    inclination_pattern = '(' + '|'.join(inclination) + ')'
    line_attr_num = Numbers(doc)
    i_t_s = r'\b{}\b.*\b{}\b'.format(inclination_pattern, size_pattern)  # inclination and then size
    if len(line_attr_num) == 2:
        if re.search(i_t_s, txt):
            length = float(line_attr_num[1])
            slope = float(line_attr_num[0])
        else:
            length = float(line_attr_num[0])
            slope = float(line_attr_num[1])
    elif len(line_attr_num) == 1:
        only_length = r'\b{}\b.*(\b{}\b).*(unknown|not given|not specified)'.format(size_pattern,
inclination_pattern)
        only_slope = r'\b{}\b.*(\b{}\b).*(unknown|not given|not specified)'.format(inclination_pattern,
size_pattern)
        not_length = r'\b{}\b.*(unknown|not given|not specified).*\b{}\b'.format(size_pattern,
inclination_pattern)
        not_slope = r'\b{}\b.*(unknown|not given|not specified).*\b{}\b'.format(inclination_pattern,
size_pattern)
        for_length = r'\b{}\b'.format(size_pattern)
        for_slope = r'\b{}\b'.format(inclination_pattern)
        if re.search(only_length, txt):
            length = float(line_attr_num[0])
        elif re.search(only_slope, txt):
            slope = float(line_attr_num[0])
        elif re.search(not_length, txt):
            slope = float(line_attr_num[0])
        elif re.search(not_slope, txt):
            length = float(line_attr_num[0])
        elif re.search(for_length, txt):
            length = float(line_attr_num[0])
        elif re.search(for_slope, txt):
            slope = float(line_attr_num[0])


    if len(tags) == 1:
        if len(tags[0]) == 1:
            p1 = tags[0]
        else:
            tag = tags[0]
    elif len(tags) == 2:
        if len(tags[0]) == 1 and len(tags[1]) == 1:
            p1, p2 = tags[0], tags[1]
        elif len(tags[0]) == 1 and len(tags[1]) >= 2:
            p1, tag = tags[0], tags[1]
    elif len(tags) == 3:
        p1, p2, tag = tags[0], tags[1], tags[2]
Edge(p1=p1, p2=p2, length=length, slope=slope, tag=tag, fill_color=fill_color, visibility=visibility)



def create_polygon(txt, doc, sh_attr):
    sh_attr_num = Numbers(doc)
    tags = re.findall(r'[A-Z]*', txt)
    tags = list(filter(lambda x: x != '', tags))
    tags = sorted(tags, key=len)
    for i in tags:
        if i in cur_points.keys():
            sh_attr['points'].append(i)
        elif i in cur_edges.keys():
            sh_attr['edges'].append(i)
        else:
            sh_attr['tag'] = i

    bs_len_li = ['(' + str1 + ')?' + ' ' + str2 for str1 in each for str2 in side]
    bs_len_ptr = '(' + '|'.join(bs_len_li) + '|base[-_]?length)'  # base_length pattern
    bs_ang_li = ['(' + str1 + ')?' + str2 for str1 in each for str2 in angle]
    bs_ang_ptr = '(' + '|'.join(bs_ang_li) + ')'  # base_angle pattern

    only_bs_len = r'\b{}\b.*(\b{}\b).*(unknown|not given|not specified)'.format(bs_len_ptr,
bs_ang_ptr)
    only_bs_ang = r'\b{}\b.*(\b{}\b).*(unknown|not given|not specified)'.format(bs_ang_ptr,
bs_len_ptr)
    not_bs_len = r'\b{}\b.*(unknown|not given|not specified).*\b{}\b'.format(bs_len_ptr,
bs_ang_ptr)
    not_bs_ang = r'\b{}\b.*(unknown|not given|not specified).*\b{}\b'.format(bs_ang_ptr,
bs_len_ptr)
```

```
for_bs_len = r'\b{}\b'.format(bs_len_ptr)
for_bs_ang = r'\b{}\b'.format(bs_ang_ptr)

size_pattern = '(' + '|'.join(size) + ')'
angle_pattern = '(' + '|'.join(angle) + ')'
line_attr_num = Numbers(doc)

only_length = r'{}.*({}).*(unknown|not given|not specified)'.format(size_pattern, angle_pattern)
only_angle = r'{}.*({}).*(unknown|not given|not specified)'.format(angle_pattern, size_pattern)
not_length = r'{}.*(unknown|not given|not specified).*{}'.format(size_pattern, angle_pattern)
not_angle = r'{}.*(unknown|not given|not specified).*{}'.format(angle_pattern, size_pattern)
for_length = r'{}'.format(size_pattern)
for_angle = r'{}'.format(angle_pattern)

ang_t_len_1 = r'\b{}.*\b{}'.format(angle_pattern, size_pattern)  # angle and then length
ang_t_len_2 = r'\b{}.*\b{}'.format(bs_ang_ptr, bs_len_ptr)  # angle and then length
len_t_ang_1 = r'\b{}.*\b{}'.format(size_pattern, angle_pattern)  # angle and then length

start_to_len = r'(.*?)\b{}'.format(size_pattern)
start_to_ang = r'(.*?)\b{}'.format(angle_pattern)
len_to_ang = r'\b{}(.*?)\b{}'.format(size_pattern, angle_pattern)
ang_to_len = r'\b{}(.*?)\b{}'.format(angle_pattern, size_pattern)
len_to_end = r'\b{}(.*?)$'.format(size_pattern)
ang_to_end = r'\b{}(.*?)$'.format(angle_pattern)

num_ptr = r'(\d+(?:\.\d+)?)(?:\s*,\s*|\s+and\s+)?'

if len(sh_attr_num) == 1:

    if re.search(only_bs_len, txt):
        sh_attr['base_length'] = float(sh_attr_num[0])
    elif re.search(only_bs_ang, txt):
        sh_attr['base_angle'] = float(sh_attr_num[0])
    elif re.search(not_bs_len, txt):
        sh_attr['base_angle'] = float(sh_attr_num[0])
    elif re.search(not_bs_ang, txt):
        sh_attr['base_length'] = float(sh_attr_num[0])
    elif re.search(for_bs_len, txt):
        sh_attr['base_length'] = float(sh_attr_num[0])
    elif re.search(for_bs_ang, txt):
        sh_attr['base_angle'] = float(sh_attr_num[0])

    elif re.search(only_length, txt):
        sh_attr['base_length'] = float(line_attr_num[0])
    elif re.search(only_angle, txt):
        sh_attr['base_angle'] = float(line_attr_num[0])
    elif re.search(not_length, txt):
        sh_attr['base_angle'] = float(line_attr_num[0])
    elif re.search(not_angle, txt):
        sh_attr['base_length'] = float(line_attr_num[0])
    elif re.search(for_length, txt):
        sh_attr['base_length'] = float(line_attr_num[0])
    elif re.search(for_angle, txt):
        sh_attr['base_angle'] = float(line_attr_num[0])

elif len(sh_attr_num) == 2:
    if re.search(ang_t_len_1, txt) or re.search(ang_t_len_2, txt):
        sh_attr['base_angle'] = float(sh_attr_num[0])
        sh_attr['base_length'] = float(sh_attr_num[1])
    else:
        sh_attr['base_length'] = float(sh_attr_num[0])
        sh_attr['base_angle'] = float(sh_attr_num[1])

elif len(sh_attr_num) > 2:
```

```
if re.search(ang_t_len_1, txt):
    sub_str = re.findall(start_to_ang, txt)
    numbers = re.findall(num_ptr, sub_str[0][0])
    if len(numbers) == 0:
        sub_str_1 = re.findall(ang_to_len, txt)
        numbers = re.findall(num_ptr, sub_str_1[0][1])
        sh_attr['angles'] = [float(i) for i in numbers]
        sub_str_2 = re.findall(len_to_end, txt)
        numbers = re.findall(num_ptr, sub_str_2[0][1])
        sh_attr['lengths'] = [float(i) for i in numbers]
    else:
        sh_attr['angles'] = [float(i) for i in numbers]
        sub_str_1 = re.findall(ang_to_len, txt)
        numbers = re.findall(num_ptr, sub_str_1[0][1])
        if len(numbers) == 0:
            sub_str_1 = re.findall(len_to_end, txt)
            numbers = re.findall(num_ptr, sub_str_1[0][1])
            sh_attr['lengths'] = [float(i) for i in numbers]
        else:
            sh_attr['lengths'] = [float(i) for i in numbers]


elif re.search(len_t_ang_1, txt):
    sub_str = re.findall(start_to_len, txt)
    numbers = re.findall(num_ptr, sub_str[0][0])
    if len(numbers) == 0:
        sub_str_1 = re.findall(len_to_ang, txt)
        numbers = re.findall(num_ptr, sub_str_1[0][1])
        sh_attr['lengths'] = [float(i) for i in numbers]
        sub_str_2 = re.findall(ang_to_end, txt)
        numbers = re.findall(num_ptr, sub_str_2[0][1])
        sh_attr['angles'] = [float(i) for i in numbers]
    else:
            sh_attr['lengths'] = [float(i) for i in numbers]
            sub_str_1 = re.findall(len_to_ang, txt)
            numbers = re.findall(num_ptr, sub_str_1[0][1])
            if len(numbers) == 0:
                sub_str_1 = re.findall(ang_to_end, txt)
                numbers = re.findall(num_ptr, sub_str_1[0][1])
                sh_attr['angles'] = [float(i) for i in numbers]
            else:
                sh_attr['angles'] = [float(i) for i in numbers]

        elif re.search(for_length, txt):
            numbers = re.findall(num_ptr, txt)
            sh_attr['lengths'] = [float(i) for i in numbers]
        elif re.search(for_angle, txt):
            numbers = re.findall(num_ptr, txt)
            sh_attr['angles'] = [float(i) for i in numbers]

    if re.search(r'equilateral triangle', txt, re.IGNORECASE):
        sh_attr['angles'] = [60, 60, 60]
    elif re.search(r'isosceles triangle', txt, re.IGNORECASE):
        if len(sh_attr['angles']) == 0 and len(sh_attr['angles_tags']) == 0:
            sh_attr['angles'] = [90, 45, 45]

    elif re.search(r'scalene triangle', txt, re.IGNORECASE):
        if len(sh_attr['angles']) == 0:
            sh_attr['angles'] = [30, 60, 90]
        elif len(sh_attr['angles']) == 1:
            sh_attr['angles'].append(randonNum((180 - sh_attr['angles'][0]) / 2, 180 -
sh_attr['angles'][0]))

    elif re.search(r'right[-\s]?(angle|angled)? triangle', txt, re.IGNORECASE):
        if 90 not in sh_attr['angles']:
            sh_attr['angles'].append(90)
    elif re.search(r'acute triangle', txt, re.IGNORECASE):
        sh_attr['angles'] = [60, 60, 60]

    elif re.search(r'obtuse triangle', txt, re.IGNORECASE):
        sh_attr['angles'] = [120, 30, 30]

    elif re.search(r'isosceles right triangle', txt, re.IGNORECASE):
        sh_attr['angles'] = [90, 45, 45]
    elif re.search(r'rectangle', txt, re.IGNORECASE):
        sh_attr['angles'] = [90, 90, 90, 90]
        Point(x=-100, y=0, tags='r1')
        Point(x=100, y=0, tags='r2')
        Point(x=100, y=100, tags='r3')
        Point(x=-100, y=100, tags='r4')
        sh_attr['points'] = ['r1', 'r2', 'r3', 'r4']

Poly(count=sh_attr['count'],
     points=sh_attr['points'],
     edges=sh_attr['edges'],
     angles_tags=sh_attr['angles_tags'],
     angles=sh_attr['angles'],
     base_edge=sh_attr['base_edge'],
     base_length=sh_attr['base_length'],
     tag=sh_attr['tag'],
     visibility=sh_attr['visibility'])


def create_object(txt, doc):
    nouns = Nouns(doc)

    if 'point' in nouns:
        create_point(txt, doc)

    # creating a line
    elif 'line' in nouns:
        create_edge(txt, doc)

    # creating a polygon
    else:
        sh_attr = {'count': None,
                   'edges': [],
                   'lengths': [],
                   'points': [],
                   'angles_tags': [],
                   'angles': [],
                   'base_length': None,
                   'base_edge': None,
                   'base_angle': None,
                   'tag': None,
                   'visibility': 1
                   }
        for i in doc:
            j = i.text.lower()
            if j in ed_3_sh:
                sh_attr['count'] = 3
                create_polygon(txt, doc, sh_attr)
            elif j in ed_4_sh:
                sh_attr['count'] = 4
                create_polygon(txt, doc, sh_attr)
            elif j in ed_5_sh:
                sh_attr['count'] = 5
                create_polygon(txt, doc, sh_attr)
            elif j in ed_6_sh:
                sh_attr['count'] = 6
                create_polygon(txt, doc, sh_attr)
            elif j in ed_7_sh:
                sh_attr['count'] = 7
                create_polygon(txt, doc, sh_attr)
            elif j in ed_8_sh:
                sh_attr['count'] = 8
                create_polygon(txt, doc, sh_attr)
            elif j in ed_9_sh:
                sh_attr['count'] = 9
                create_polygon(txt, doc, sh_attr)
            elif j in ed_10_sh:
                sh_attr['count'] = 10
                create_polygon(txt, doc, sh_attr)


def prompt(txt):
    des_all_obj_btns()
    Point(x=0, y=-500, tags='y1')
    Point(x=0, y=500, tags='y2')
    Point(x=-500, y=0, tags='x1')
Point(x=500, y=0, tags='x2')
    Edge(p1='x1', p2='x2', tag='x-axis', fill_color='red', width=3)
    Edge(p1='y1', p2='y2', tag='y-axis', fill_color='red', width=3)
    showEdge('x-axis')
    showEdge('y-axis')

    doc = nlp(txt)
    verbs = Verbs(doc)
    global create
    for i in verbs:
        if i in create:
            create_object(txt, doc)
            break
    if re.search(r'clear canvas', txt, re.IGNORECASE):
        print('clearing canvas')
        canvas.delete('all')
        reset()

    drawAllPoints()
    drawAllEdges()



# Get the size of the canvas
WIDTH = window.winfo_screenwidth()
HEIGHT = window.winfo_screenheight()

# create interface elements in tkinter


label = tk.Label(window, text="LIVE LECTURE VISUAL GENERATION", height=1, width=WIDTH)
label.config(font=("Arial", 14, "bold"), fg='#FFD700', bg='#000080')
label.place(x=0, y=0)
label.config(highlightthickness=1, highlightbackground="black")
label.pack()

text_div = tk.Frame(window, bd=1, relief="solid", width=WIDTH, height=85)
text_div.pack(anchor='nw')
text_div.config(borderwidth=1, relief="solid", bg="#191970")

input_field = tk.Text(text_div, bd=1, relief="solid", width=140, height=4)
input_field.pack(side='left', anchor='nw')
input_field.config(bg="#E2FEFE", fg="#505400", font=("Arial", 12, "bold"), insertbackground="red")

# Create a button in the division
```

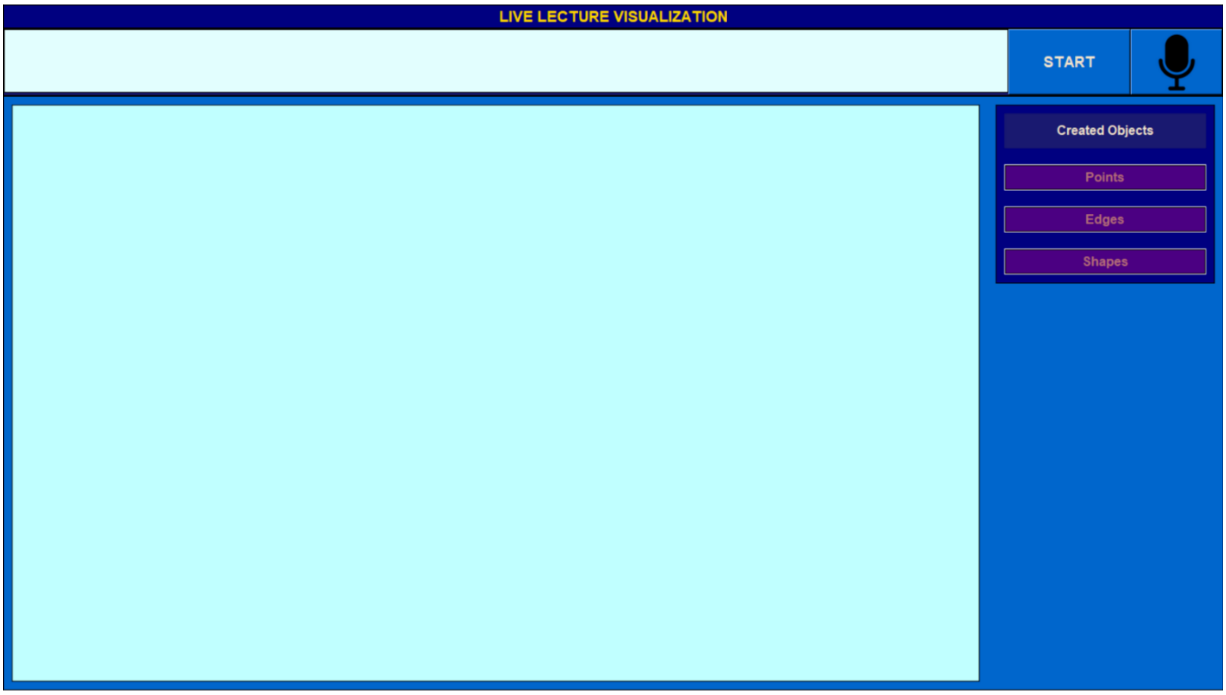
```
submit_button = tk.Button(text_div, text="START", width=12, height=3, command=lambda: append_lecture())
submit_button.config(font=("Arial", 14, "bold"), bg='#0066CC', fg='#F7E7CE')
submit_button.pack(side='left', anchor='nw')

# Load the microphone icon image
mic_image = Image.open("mic.png")  # Replace "mic.png" with your own image file
mic_image = mic_image.resize((80, 80), resample=Image.Resampling.LANCZOS)
mic_icon = ImageTk.PhotoImage(mic_image)

# Create a label and an image for the microphone icon in the division
mic_label = tk.Button(text_div, image=mic_icon, width=140, height=77, command=lambda: toggle_listening(input_field))
mic_label.pack(side='left', anchor='nw')
mic_label.config(bg='#0066CC')

# canvas and side tools
# canvas and side tools

cvs_tool_div = tk.Frame(window, bd=1, relief="solid")
cvs_tool_div.pack(anchor='nw')
cvs_tool_div.config(borderwidth=1, relief="solid", bg="#0066CC")

cvs_width = WIDTH - 300
cvs_height = HEIGHT - 120

canvas = tk.Canvas(cvs_tool_div, width=cvs_width - 20, height=cvs_height - 20, bg="#C0FFFF")

# change the canvas as origin based
canvas.configure(scrollregion=(-(cvs_width / 2), -cvs_height / 2, cvs_width / 2, cvs_height / 2))
canvas.pack(side='left', padx=10, pady=10)
canvas.config(highlightthickness=1, highlightbackground="black")
```

```
tool_div = tk.Frame(cvs_tool_div, width=280, height=cvs_height - 20, bg='#000080', bd=1,
relief="solid")
tool_div.pack(side='top', padx=10, pady=10)
tool_div.config(borderwidth=1, relief="solid")

# Add a label inside the tool_div frame
objects_label = tk.Label(tool_div, text="Created Objects", height=2, width=27)
objects_label.config(font=("Arial", 12, "bold"), bg='#191970', fg='#F7E7CE')
objects_label.pack(padx=10, pady=10, anchor='n')

point_objects_div = tk.Frame(tool_div, height=100, width=280)
point_objects_div.pack(padx=10, pady=10, side='top', anchor='n')

edge_objects_div = tk.Frame(tool_div, height=100, width=280)
edge_objects_div.pack(padx=10, pady=10, side='top', anchor='n')

sh_objects_div = tk.Frame(tool_div, height=100, width=280)
sh_objects_div.pack(padx=10, pady=10, side='top', anchor='n')

point_objects_btn = tk.Button(point_objects_div, text='Points', height=1, width=35,
                    command=lambda: show_objects('points'))
point_objects_btn.config(font=("Arial", 12, "bold"), borderwidth=1, relief="solid", bg='#4B0082',
fg='#B76E79')
point_objects_btn.pack(padx=1, pady=1, side='top', anchor='n')

edge_objects_btn = tk.Button(edge_objects_div, text='Edges', height=1, width=35,
                   command=lambda: show_objects('edges'))
edge_objects_btn.config(font=("Arial", 12, "bold"), borderwidth=1, relief="solid", bg='#4B0082',
fg='#B76E79')
edge_objects_btn.pack(padx=1, pady=1, side='top', anchor='n')
sh_objects_btn = tk.Button(sh_objects_div, text='Shapes', height=1, width=35,
                  command=lambda: show_objects('shapes'))
sh_objects_btn.config(font=("Arial", 12, "bold"), borderwidth=1, relief="solid", bg='#4B0082',
fg='#B76E79')
```

```
sh_objects_btn.pack(padx=1, pady=1, side='top', anchor='n')

# calling the animate function

animateCanvas()

# looping the tkinter(interface) window

window.mainloop()
```

CHAPTER 6

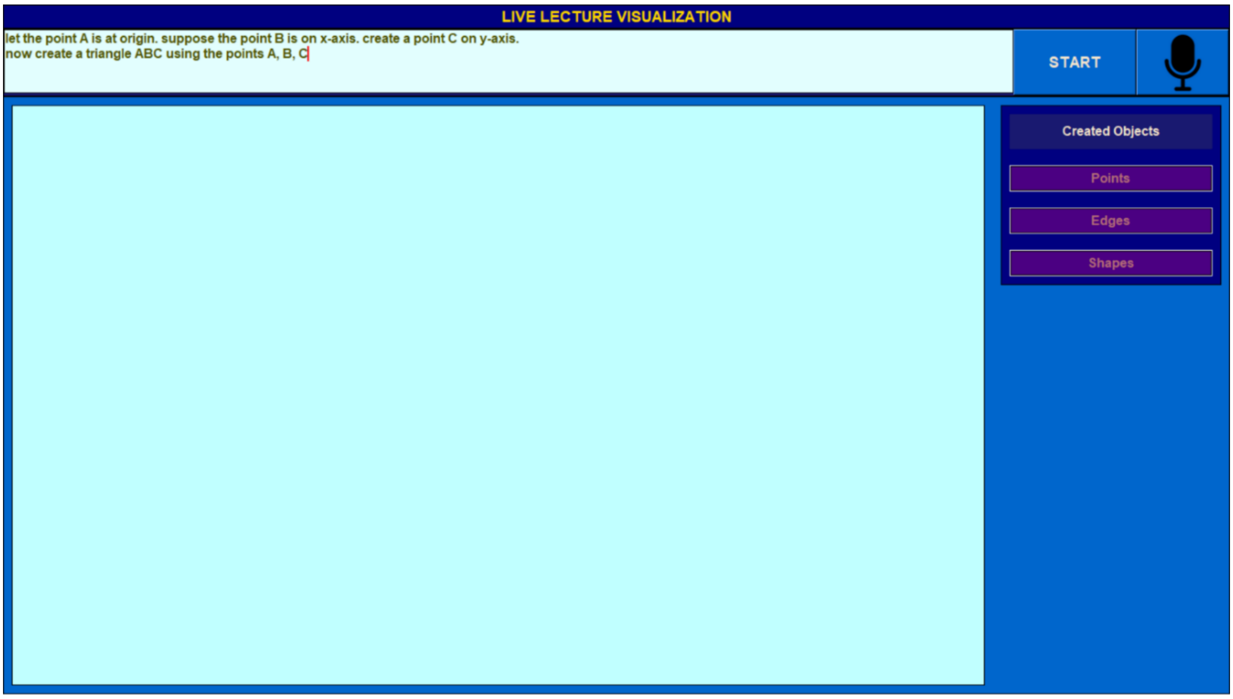
INTERFACE



CHAPTER 7

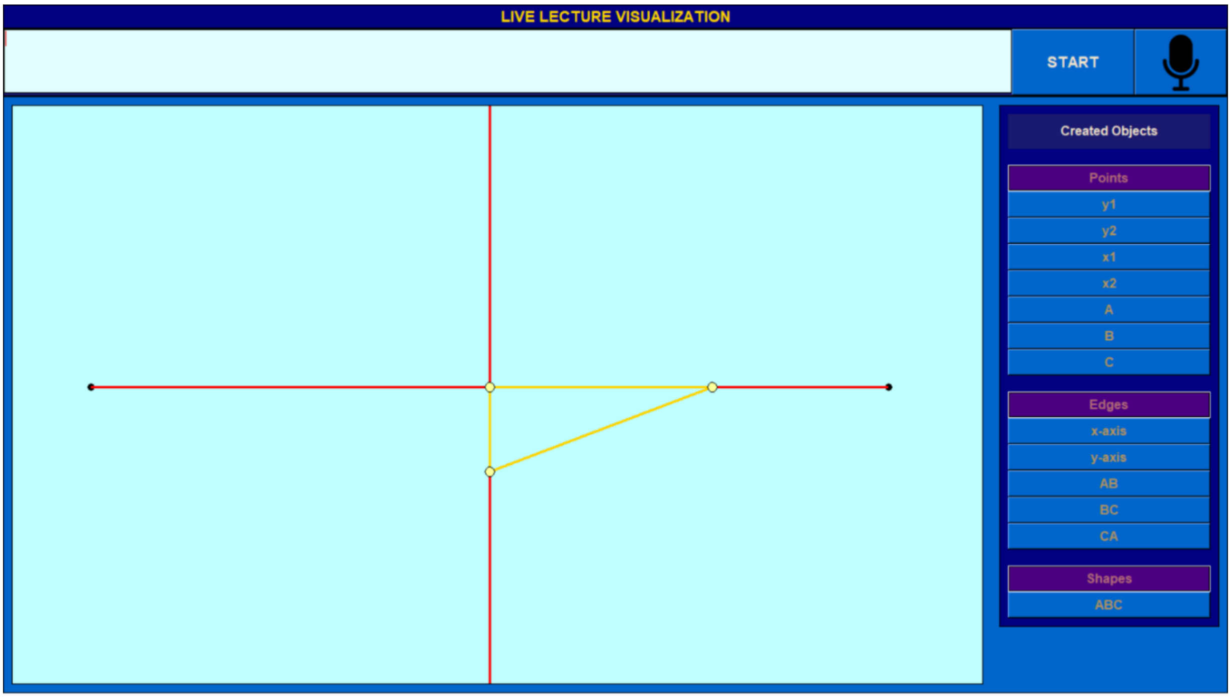
OUTPUT:

7.1. Text Input:



let the point A is at origin. suppose the point B is on x-axis. create a point C on y-axis.
now create a triangle ABC using the points A, B, C

7.2. Expected Output:



CHAPTER 8

FEATURES OF NEW SYSTEM:

The proposed system is designed in accordance with user requirements to fulfil almost all of them.

8.1. User Friendly:

The GUI provided in the proposed system is clean and can be accessed easily.

8.2. Results can be generated in Real Time:

The system can produce the Results in the Real time by processing the speech of the lecturer and generate the relevant visuals which can increase the effectiveness of students learning.

CHAPTER 9

NON-FUNCTIONAL REQUIREMENTS:

9.1. Performance:

- The assistant shall provide real-time visual generation with minimal processing delays.

- The system shall handle a large number of concurrent users without significant performance degradation.

- The response time for voice recognition and visual generation should be within acceptable limits.

9.2. Usability:

- The system shall have an intuitive and user-friendly interface for ease of use.
- The system shall provide clear instructions and guidance for both lecturers and students.

CHAPTER 10

RESULTS:

The project meets all the defined requirements and procedures that are mentioned in the Software Requirement Specification, and it suggests the usage of this type of model in ICT based Education for increasing learning efficiency of the students

CHAPTER 11

CONCLUSION:

We conclude that this project addresses the problems of the existing system, and it can be used to generate visual using speech recognition and text. It addresses the problems faces by the students in the traditional lectures.

CHAPTER 12

FUTURE SCOPE:

The system is currently developed for only a small domain called the geometry. However, the system can be used in a variety of ways in different phases such as:

- We can use this procedure to develop a system that can understand multiple domains.
- Students can use this for self-study.
- Using the Image processing can help in producing more dynamic visuals in all domains.

REFERENCES:

1. Research Papers:

- https://arxiv.org/abs/2205.11487
- https://arxiv.org/abs/1406.2661
- https://www.researchgate.net/publication/359441889_Text_to_Image_using_Deep_Learning

2. Existing Applications:

- **Dall-E**
  https://labs.openai.com/

- **DeepAI**
  https://deepai.org/machine-learning-model/text2img

- **Dream Studio**
  https://beta.dreamstudio.ai/generate

- **Replicate**
  https://replicate.com/stability-ai/stable-diffusion

3. Articles and Resources:

- https://medium.com/mlearning-ai/10-best-free-to-use-text-to-image-generators-25743b3a5d50#17b3
- https://en.wikipedia.org/wiki/Text-to-image_model#:~:text=A%20text%2Dto%2Dimage%20model,advances%20in%20deep%20neural%20networks
- https://www.w3schools.com/python/python_regex.asp

4. Reference Videos:

- https://www.youtube.com/playlist?list=PLzgPDYo_3xumT2sfELR4_YV3aojaxkUC9
- https://www.youtube.com/playlist?list=PLCC34OHNcOtoC6GglhF3ncJ5rLwQrLGnV
- https://www.youtube.com/playlist?list=PLc2rvfiptPSSS-iwKS_lxI3MZr8Mbi4Zu
- https://youtu.be/SVcsDDABEkM
- https://youtu.be/7xc0Fs3fpCg