My Expirence Animating with Manim

Michael Xie
Jack Miner Public School
Grade 8

April 13, 2025

1 Introduction

Manim is a powerful animation engine that allows you to create high-quality animations for mathematical concepts. It is widely used in educational videos and presentations. In this article, I will share my experience using Manim to create animations for my class project. I will discuss the challenges I faced, the solutions I found, and the final product I created.

2 Getting Started

I already had a Manim environment set up on my computer from a previous project, along with a solid understanding of Python. To begin, I researched my topic, holomorphic dynamics, by reading several articles and watching videos, including 3Blue1Brown's excellent explanation on the subject. This gave me a clear idea of what a Manim animation should look like.

I started coding by using Matplotlib to create a static plot of the holomorphic function I was working with—the Mandelbrot set. After a few rounds of debugging, I successfully wrote a function that generates an ndarray representing the Mandelbrot set, along with several additional utility functions. These included:

1. `mandelbrot`: Generate a Mandelbrot set image.
   This function computes the Mandelbrot set for a given range of real and imaginary values, and returns a 2D array representing the fractal image. The computation is optimized by skipping points inside the main cardioid and the period-2 bulb.
   Parameters:
   - `rmin` (float): The minimum value of the real axis.
   - `rmax` (float): The maximum value of the real axis.
   - `cmax` (complex): The maximum value of the imaginary axis (only the imaginary part is used).
   - `width` (int): The width of the output image in pixels.
   - `height` (int): The height of the output image in pixels. If odd, it will be adjusted to the next even number.
   - `maxiter` (int): The maximum number of iterations to determine divergence.

   Returns:
   - `numpy.ndarray`: A 2D array of shape (`height`, `width`) containing the Mandelbrot set. Each value represents the iteration count at which the point diverged, or 0 if the point is in the set.
2. `mandelbrot_point_cloud`: Generate the orbit for a single point.
   This function computes the orbit of a single point in the Mandelbrot set and returns a NDarray of complex numbers representing the orbit.
   Parameters:
   - `c` (complex): The complex number to evaluate in the Mandelbrot set.
   - `maxiter` (int): The maximum number of iterations to perform.

   Returns:
   - `numpy.ndarray:` A 2D array of shape (maxiter, 2), where each row contains the real and imaginary parts of the complex number at each iteration.
3. `mandelbrot_from_center`: Generate a Mandelbrot set image centered at a specific point with a zoom level.
   This function computes the Mandelbrot set for a given center point and zoom level, allowing for detailed exploration of the fractal.
   Parameters:
   - `centerpoint` (complex): The center of the viewing field.
   - `zoom` (float): The zoom level (higher values zoom in).
   - `width` (int): The width of the output image in pixels.

    - `height` (int): The height of the output image in pixels.
    - `maxiter` (int): The maximum number of iterations to determine divergence.

   Returns:

    - `numpy.ndarray`: A 2D array of shape (`height`, `width`) containing the Mandelbrot set.

4. `multibrot`: Generate a generalized Mandelbrot set with a custom exponent.
   This function computes the fractal for a given exponent, creating variations of the Mandelbrot set.
   Parameters:

    - `rmin`, `rmax` (float): The range of the real axis.
    - `cmax` (complex): The maximum value of the imaginary axis.
    - `width`, `height` (int): The dimensions of the output image in pixels.
    - `maxiter` (int): The maximum number of iterations.
    - `exponent` (int): The exponent used in the fractal formula.

   Returns:

    - `numpy.ndarray`: A 2D array representing the fractal.

5. `julia`: Generate a Julia set for a given complex parameter.
   This function computes the Julia set, a related fractal to the Mandelbrot set, for a specific complex parameter.
   Parameters:

    - `c` (complex): The complex parameter for the Julia set.
    - `width`, `height` (int): The dimensions of the output image in pixels.
    - `maxiter` (int): The maximum number of iterations.

   Returns:

    - `numpy.ndarray`: A 2D array representing the Julia set.

6. `multicorn`: Generate a Multicorn fractal with a custom exponent.
   This function computes the Multicorn fractal, a variation of the Mandelbrot set with conjugation.
   Parameters:

    - `rmin`, `rmax` (float): The range of the real axis.
    - `cmax` (complex): The maximum value of the imaginary axis.
    - `width`, `height` (int): The dimensions of the output image in pixels.
    - `maxiter` (int): The maximum number of iterations.
    - `exponent` (int): The exponent used in the fractal formula.

   Returns:

    - `numpy.ndarray`: A 2D array representing the Multicorn fractal.

7. `buhdabrot`: Generate a Buhdabrot fractal using a 4D histogram.
   This function computes the Buhdabrot fractal by sampling random points and tracking their trajectories in a 4D histogram.
   Parameters:

    - `rmin`, `rmax` (float): The range of the real axis.
    - `cmax` (complex): The maximum value of the imaginary axis.
    - `width`, `height` (int): The dimensions of the output image in pixels.
    - `maxiter` (int): The maximum number of iterations.
    - `sample_size` (int): The number of random samples to generate.

   Returns:

    - `numpy.ndarray`: A 4D histogram representing the Buhdabrot fractal.

3 Creating the Animations

Once I completed the functions, I began programming the Manim animations. Rather than following a strict order, I worked on the scenes that made the most sense at the time, guided by a general vision of the final product. I started by rendering a static Mandelbrot set and tracing the orbit of a point. To achieve this, I used `ComplexValueTracker`s to animate the changing point and equation text, while applying the inferno colormap for visualization. **Here is the code for the animations**

```python
class MandelbrotScene(VoiceoverScene):
    def construct(self):
        # Set up speech service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        # Mandelbrot set parameters
        rmin, rmax = -2, 1
        cmax = 1.5 + 1j
        width, height = 3840, 2160
        maxiter = 256
        colormap = cm.get_cmap("inferno")

        # Generate Mandelbrot set image
        mandelbrot_set = mandelbrot(rmin, rmax, cmax, width, height, maxiter)
        normalized_set = np.log(1 + mandelbrot_set) / np.log(3) / np.log(maxiter)
        colored_image = (colormap(normalized_set)[:, :, :3] * 255).astype(np.uint8)
        mandelbrot_mobject = ImageMobject(colored_image, scale_to_resolution=2160)

        # Display Mandelbrot set
        with self.voiceover(text="Lets see how the orbits of a couple points appear " \
        "on the full set."):
            self.add(mandelbrot_mobject)
            self.wait(5)

        # Animate points on the Mandelbrot set
        c = ComplexValueTracker(0 + 0j)
        dots = VGroup()
        lines = VGroup()
        text = MathTex(r"f_c(z)= z^2+c").to_corner(UL)

        # Update dots to represent orbit points
        dots.add_updater(lambda x: x.become(
            VGroup(*[Dot(np.array([point[0], point[1], 0]), radius=0.05, color=WHITE)
                     for point in mandelbrot_point_cloud(c.get_value(), maxiter)])))

        # Update lines to connect orbit points
        lines.add_updater(lambda x: x.become(
            VGroup(*[
                Line(
                    np.array([point[0][0], point[0][1], 0]),
                    np.array([point[1][0], point[1][1], 0]),
                    color=BLUE,
                    stroke_width=1
                )
                for point in zip(mandelbrot_point_cloud(c.get_value(), maxiter)[:-1],
                                 mandelbrot_point_cloud(c.get_value(), maxiter)[1:])
            ])
        ))

        # Update the formula text
        text.add_updater(lambda x: x.become(
            MathTex(f"f_{{{c.get_value().real:.2f} {'+' if c.get_value().imag >= 0 else '-'}" \
                    f"{abs(c.get_value().imag):.2f}i}}(z) = z^2 {'+' if c.get_value().real >= 0 else '-'}" \
                    f"{abs(c.get_value().real):.2f} {'+' if c.get_value().imag >= 0 else '-'}" \
                    f" {abs(c.get_value().imag):.2f}").to_corner(UL)
        ))

        self.add(dots, lines, text)

        # Animate transitions between different points
        points = [
            0.355 + 0.355j,
            -0.10109636384562 + 0.95628651080914j,
            0 + 1j,
            -0.1 + 0.75j,
            0 + 0j,
            -0.335 + 0.335j
        ]
        for point in points:
                self.play(c.animate.set_value(point), run_time=3)
                self.wait(2)

        # Cleanup
        self.play(FadeOut(dots, lines, text))
        self.wait(2)
```

Main.py

The next scene I created highlighted the orbit of a single point in the Mandelbrot set on a `ComplexPlane`. I used a point cloud to visualize the orbit of the point and arrows to indicate transitions between points. Additionally, I utilized a `ComplexValueTracker` to animate the movement of the point and a `MathTex` object to dynamically display the equation representing the point. **This is the code for the scene:**

```python
class OnePointExample(VoiceoverScene):
    def construct(self):
        # Set up speech service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        # Display formula in the upper-left corner
        text = MathTex(r"{z_0}^2 + c =", r"z_1").to_corner(UL)
        self.play(Write(ComplexPlane().add_coordinates()), run_time=2)
        self.play(Write(text), run_time=2)

        # Define orbit points for a specific complex number
        orbit_points = [
            -0.1 + 0.75j,
            (-0.1 + 0.75j) ** 2 + (-0.1 + 0.75j),
            ((-0.1 + 0.75j) ** 2 + (-0.1 + 0.75j)) ** 2 + (-0.1 + 0.75j),
        ]

        # Animate the orbit points
        point = ComplexValueTracker(orbit_points[0])
        dot = Dot(
            np.array([point.get_value().real, point.get_value().imag, 0]),
            radius=0.05,
            color=WHITE,
        )
        dot.add_updater(lambda x: x.move_to(np.array([point.get_value().real,
                                                       point.get_value().imag, 0])))
        self.play(FadeIn(dot), run_time=0.3)

        # Add voiceover for the explanation
        with self.voiceover(text="Let's visualize the orbit of a single point under" \
        " the iteration of our function."):
            self.wait(2)

        # Iterate through the orbit points
        for _ in range(2):  # Repeat the cycle twice
            for i in range(len(orbit_points)):
                if i > 0:
                    # Draw an arrow between points
                    arrow = CurvedArrow(
                        np.array([orbit_points[i - 1].real, orbit_points[i - 1].imag, 0]),
                        np.array([orbit_points[i].real, orbit_points[i].imag, 0]),
                        color=BLUE,
                        tip_length=0.2
                    )
                    self.play(Write(arrow), run_time=0.5)
                    self.wait(0.75)
                    self.play(FadeOut(arrow), run_time=0.5)

                # Update the dot and formula text
                self.play(point.animate.set_value(orbit_points[i]))
                new_text = MathTex(
                    f"({orbit_points[i]:.2f})^2 + c = {orbit_points[i]**2 + orbit_points[i]:.2f}"
                ).to_corner(UL)
                self.play(Transform(text, new_text))

            # Complete the cycle with an arrow back to the first point
            arrow = CurvedArrow(
                np.array([orbit_points[-1].real, orbit_points[-1].imag, 0]),
                np.array([orbit_points[0].real, orbit_points[0].imag, 0]),
                color=BLUE,
                tip_length=0.2
            )
            self.play(Write(arrow), run_time=0.5)
            self.wait(0.75)
            self.play(FadeOut(arrow), run_time=0.5)
            self.play(point.animate.set_value(orbit_points[0]))

        # Cleanup
        self.play(FadeOut(text), FadeOut(dot), run_time=0.5)
        dot.clear_updaters()
        self.wait(2)
```

Main.py

After animating the oribit of a single point, I deciding to animate the orbit of multiple points, once agian using `ComplexValueTra` to animate the `Dot` and the `MathTex` objects, as well as `mandelbrot_point_cloud` function to generate the orbits. it was also, once agian on the background of a `ComplexPlane`. **Here is the code for the scene:**

```python
class MultiplePointsExample(VoiceoverScene):
    def construct(self):
        # Set up speech service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        self.play(Write(ComplexPlane()))

        Dots = VGroup()
        lines = VGroup()
        text = MathTex(r"f_c(z) = z^2 + c")
        c = ComplexValueTracker(0 + 0j)

        Dots.add_updater(lambda x: x.become(VGroup(*[Dot(np.array([i[0], i[1], 0]),
                                                          radius=0.02) for i in mandelbrot_point_cloud(c.get_value(), 25)]))))
        lines.add_updater(lambda x: x.become(
            VGroup(*[
                Line(
                    np.array([point[0][0], point[0][1], 0]),
                    np.array([point[1][0], point[1][1], 0]),
                    color=BLUE,
                    stroke_width=1
                )
                for point in zip(mandelbrot_point_cloud(c.get_value(), 25)[:-1],
                                 mandelbrot_point_cloud(c.get_value(), 25)[1:])
            ])
        ))
        text.add_updater(lambda x: x.become(MathTex(f"f_{{{c.get_value():.2f}}}(z) = z^2 {'+' if c.get_value().real >= 0 else '-'}" \
                                                    f"{abs(c.get_value()).real:.2f} {'+' if c.get_value().imag >= 0 else '-'}" \
                                                    f"{abs(c.get_value().imag):.2f}i").to_corner(UL)))

        self.add(Dots, lines)
        with self.voiceover(text="Here is the visualization of multiple points iterating under our dynamic system, with thier orbits traced."):
            self.play(Write(text))

        points = [
            (0.335 - 0.335j, "Some points fall into stable cycles;"),
            (-0.2 - 0.3j, "Other points aproach a attracting limit point;"),
            (0.33 + 0.06j, " Some points are attracted to a repelling limit point;"),
            (0.5 + 0.5j, "And some points just shoot of to infinity;"),
            (0, "Or at some points, like the origin, the function is it's own inverse!")
        ]

        for point, narration in points:
            with self.voiceover(text=narration) as tracker:
                self.play(c.animate.set_value(point), run_time=tracker.duration)
                self.wait(3)

        self.play(FadeOut(Dots, lines), run_time=3)
```

Main.py

The next scene I worked on was a text-based scene that explained the concept of the Mandelbrot set. I used a `MathTex` object to display the equations, and a `Text` object to display the text. **The code for the scene is as follows:**

```python
class TextScene(VoiceoverScene):
    def construct(self):
        # Set up speach service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        # Title
        title = Text("What is the Mandelbrot Set?", font_size=48).to_edge(UP)
        with self.voiceover(text="What is the Mandelbrot Set?"):
            self.play(Write(title), run_time=2)

        # Description
        description = Tex(
            "The Mandelbrot Set is a set of complex numbers c\n"
            "for which the function $z = z^2 + c$ does not diverge\n"
            "when iterated from $z = 0$.",
            font_size=36,
        ).next_to(title, DOWN, buff=0.5)
        with self.voiceover(text="The Mandelbrot Set is a set of complex numbers c " \
        "for which the function z equals z squared plus c does not diverge when iterated from z equals zero."):
            self.play(Write(description), run_time=4)
        self.wait(5)

        # Transition to example iterations
        with self.voiceover(text="Now, let's look at some example iterations."):
            self.play(FadeOut(title), FadeOut(description))
        example_title = Text("Example Iterations", font_size=42).to_edge(UP)
        self.play(Write(example_title), run_time=2)

        # Example iterations for a specific complex number
        c = 0.355 + 0.355j
        text_arr = [i for i in mandelbrot_point_cloud(c, 10)]
        example_text = [
            MathTex(
            f"z_{{{i+1}}}", "=", f"({text_arr[i][0]:.2f}",
            f"{'+' if text_arr[i][1] >= 0 else '-'}",
            f"{abs(text_arr[i][1].imag):.2f}i)", f"{'+' if c.real >= 0 else '-'}",
            f"({abs(c.real):.2f}", f"{'+' if c.imag >= 0 else '-'}",
            f"{abs(c.imag):.2f}i)"
            ).set_color_by_tex_to_color_map({
            f"z_{{{i+1}}}": BLUE,
            f"({text_arr[i][0]:.2f}": YELLOW,
            f"{abs(text_arr[i][1].imag):.2f}i)": YELLOW,
            f"({abs(c.real):.2f}": RED,
            f"{abs(c.imag):.2f}i)": RED
            })
            for i in range(len(text_arr) - 1)
        ]

        # Display example iterations
        example_text_mobjects = VGroup(example_text
        ).arrange(DOWN, buff=0.5).next_to(example_title, DOWN, buff=0.5)

        with self.voiceover(text="Here are the first few iterations for a specific complex number."):
            self.play(Write(example_text_mobjects[0:5]), run_time=3)
        self.wait()

        # Animate the remaining iterations
        for i in range(5, len(example_text_mobjects)):
            self.play(FadeOut(example_text_mobjects[i-5]))
            example_text_mobjects.shift(UP)
            self.play(Write(example_text_mobjects[i]), run_time=2)
            self.wait()

        self.wait(2)
        with self.voiceover(text="If the sequence remains bounded, c is in the Mandelbrot Set."):
            self.play(Unwrite(example_title))
        self.wait(2)
```

Main.py

After that, I Made a scene explaining the concept of Julia sets. The scene uses the `julia` function to generate the Julia set for a specific complex number. I used a `ComplexValueTracker` to animate changing the value of c and the inferno colormap to color the image. **Here is the code for the scene:**

```python
class JuliaSetScene(VoiceoverScene):
    def construct(self):
        # Set up speech service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        # Title
        title = Text("What are Julia Sets?", font_size=48).to_edge(UP)
        with self.voiceover(text="What are Julia Sets?"):
            self.play(Write(title), run_time=2)

        # Description
        description = Tex(
            "Julia Sets are fractals generated by iterating the function\n"
            "$z = z^2 + c$ for a fixed complex number $c$, starting from\n"
            "different initial values of $z$.",
            font_size=36,
        ).next_to(title, DOWN, buff=0.5)
        with self.voiceover(text="Julia Sets are fractals generated by iterating the function z" \
            " equals z squared plus c for a fixed complex number c, starting from different initial values of z
    ."):
            self.play(Write(description), run_time=4)
        self.wait(5)

        relationship = Text(
            "The Mandelbrot Set acts as a map for Julia Sets:\n"
            "each point in the Mandelbrot Set corresponds to a\n"
            "unique Julia Set. Points inside the Mandelbrot Set\n"
            "produce connected Julia Sets, while points outside\n"
            "produce infinitely disconnected (dust-like) Julia Sets.",
            font_size=36,
            color=BLUE
        ).next_to(description, DOWN, buff=0.5)
        with self.voiceover(text="The Mandelbrot Set acts as a map for Julia Sets. Each point in the" \
        " Mandelbrot Set corresponds to a unique Julia Set. Points inside the Mandelbrot Set produce
    connected Julia Sets," \
        " while points outside produce infinitely disconnected, dust-like Julia Sets."):
            self.play(Write(relationship), run_time=6)
        self.wait(3)

        # Transition to example
        with self.voiceover(text="Now, let's look at an example of a Julia Set."):
            self.play(FadeOut(title), FadeOut(description))
        example_title = Text("Example Julia Set", font_size=42).to_edge(UP)
        self.play(Write(example_title), run_time=2)

        self.wait(2)

        # Generate initial Julia set image
        c = ComplexValueTracker(-0.8 + 0.156j)  # Example constant for Julia set
        colormap = cm.get_cmap("inferno")
        julia_set = julia(c.get_value(), 2160, 2160, 256)
        normalized_set = np.log(1 + julia_set) / np.log(3) / np.log(256)
        normalized_set = (normalized_set - normalized_set.min()) / (normalized_set.max() - normalized_set.
    min())
        colored_image = (colormap(normalized_set)[:, :, :3] * 255).astype(np.uint8)
        julia_mobject = ImageMobject(colored_image, scale_to_resolution=1080)

        # Add updater to update the Julia set dynamically
        def update_julia(mobject):
            julia_set = julia(c.get_value(), 2160, 2160, 256)
            normalized_set = np.log(1 + julia_set) / np.log(3) / np.log(256)
            colored_image = (colormap(normalized_set)[:, :, :3] * 255).astype(np.uint8)
            mobject.become(ImageMobject(colored_image))

        julia_mobject.add_updater(update_julia)

        # Display Julia set
        with self.voiceover(text="Here is an example of a Julia Set for a specific value of c."):
            self.play(FadeIn(julia_mobject), run_time=2)
        with self.voiceover(text="Now, let's explore how the Julia Set changes as we vary the value of c.")
    :
            self.play(c.animate.set_value(0.355 - 0.335j), run_time=10)
            self.wait(2)
            self.play(c.animate.set_value(0.36 + 1j), run_time=10)
            self.wait(2)
            self.play(c.animate.set_value(0.33 + 0.06j), run_time=10)
            self.wait(2)
            self.play(c.animate.set_value(0 + 0j), run_time=10)
            self.wait(2)

        # Cleanup
        julia_mobject.clear_updaters()
        self.play(FadeOut(julia_mobject), FadeOut(example_title))
```

Main.py

Then I tried to make a scene explaining how the Mandelbrot set acts as a map for Julia sets. I used the `julia` function and loops to generate the Julia sets for each point ina grid. this was very difficult. Then, I used a updater to animate increasing the reslution of the grid while deacreasing the reslution of each julia set, showing how the Mandelbrot set acts as a map for Julia sets. however, I was not able to get this to work properly, as my grid logic was incorrect. **Here is the code for the scene:**

```
class AlternateDefintionWithJulia(VoiceoverScene):
    def construct(self):
        # Set up speach service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        colormap = cm.get_cmap('inferno')

        julia_wdith, julia_height = ValueTracker(80), ValueTracker(80)

        def normalize_color(c):
            julia_set = julia(c, int(julia_wdith.get_value() // 1), int(julia_height.get_value() // 1),
    256)
            normalized_set = np.log(1 + julia_set) / np.log(3) / np.log(256)
            return (colormap(normalized_set)[:, :, :3] * 255).astype(np.uint8)

        # Correctly cast width and height to integers
        width = int(40 * 80 // julia_wdith.get_value())
        height = int(30 * 80 // julia_height.get_value())

        # Create a grid of complex numbers
        real = np.linspace(-2, 1, width)
        imag = np.linspace(-1.5, 1.5, height)
        real_grid, imag_grid = np.meshgrid(real, imag)

        # Initialize the grid of ImageMobject instances
```

```python
        julia_images = [
            ImageMobject(normalize_color(complex(real_grid[i, j], imag_grid[i, j])), scale_to_resolution=1080).move_to(
                np.array([real_grid[i, j], imag_grid[i, j], 0])
            )
            for i in range(real_grid.shape[0]) for j in range(real_grid.shape[1])
        ]

        # Add all images to the scene
        for image in julia_images:
            self.add(image)

        # Define an updater to update the images dynamically
        def update_julia_images():
            for i in range(real_grid.shape[0]):
                for j in range(real_grid.shape[1]):
                    index = i * real_grid.shape[1] + j
                    julia_images[index].set_image(normalize_color(complex(real_grid[i, j], imag_grid[i, j])))

        # Add the updater to the scene
        self.add_updater(lambda _: update_julia_images())

        # Display the combined image
        with self.voiceover(text="Here, we see a grid of Julia Sets," \
        " each corresponding to a different point in the complex plane."):
            pass
        with self.voiceover(text="Observe how the Image changes as we increase the resolution of the grid."):
            self.play(julia_wdith.animate.set_value(5), julia_wdith.animate.set_value(5), run_time=30, rate_func=double_smooth)
        with self.voiceover(text="Now, if we zoom in on only the center pixel of the grid," \
        " we can see how the mandelbrot set acts as a map for the Julia Sets."):
            self.play(julia_wdith.animate.set_value(1), julia_wdith.animate.set_value(1), run_time=30, rate_func=double_smooth)

        with self.voiceover(text="for a point p, if the julia set of z^2 + p is connected, then p is in the mandelbrot set."):
            self.play(FadeOut(*julia_images), run_time=2)
```

Main.py

The hardest scene I made explored some generalizations of the Mandelbrot set, including the Multibrots and Buhdagrams. after countless hours of work and many rounds of debugging, I was unfortunately unable to get the scene to work properly. I used the `multibrot` and `buhdabrot` functions to generate the images, and a `ValueTracker` to animate the changing exponent. I used the inferno colormap to color the multibrot, a false color method the color the buhdagram. **Here is the code for the scene:**

```python
class Generalizations(VoiceoverScene):

    def construct(self):
        # Set up speach service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        # Title
        Title = Tex("Generalizations of the Mandelbrot Set", font_size=48).to_edge(UP)

        contents = Tex(
            r"\begin{enumerate}"
            r"\item \textbf{Multibrot Sets:} A generalization of the Mandelbrot set where the exponent" \
            r" in the iteration formula $z = z^d + c$ is replaced with a higher degree $d$."
            r"\item \textbf{Tricorn:} A fractal similar to the Mandelbrot set, but generated using" \
            r" the iteration formula $z = \overline{z}^2 + c$, where $\overline{z}$ is the complex conjugate of $z$."
            r"\item \textbf{4D Mandelbrot:} A higher-dimensional generalization of the Mandelbrot set," \
            r" visualized by projecting 4D fractals into 3D or 2D spaces."
            r"\end{enumerate}",
            font_size=36
        ).next_to(Title, DOWN, buff=0.5)

        # >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Multibrot set <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        Multibrot_title = Tex(r"1. \quad Multibrot Sets", font_size = 48).to_edge(UP)

        rmin, rmax = -2, 1
        cmax = 1.5 + 1j
        width, height = 3840, 2160
        maxiter = 256
        colormap = cm.get_cmap("inferno")
        exp = ValueTracker(0)
        exp_indicator = MathTex("d =", "0.00", substrings_to_isolate='d').set_color_by_tex('d', RED).to_corner(UL)
        exp_indicator.add_updater(lambda x: x[1].become(MathTex(f"d = {exp.get_value()}",
                                                                color = BLUE).set_color_by_tex('d', RED).to_corner(UL)))

        # Create a blank image with dimensions 3840x2160
        from PIL import Image
        blank_image = np.zeros((2160, 3840, 3), dtype=np.uint8)
        multibrot_image = ImageMobject(Image.fromarray(blank_image))

        self.add(multibrot_image)

        def Update_multibrot(old_multibrot):
            multibrot_set = multibrot(rmin, rmax, cmax, width, height, maxiter, exp.get_value())
            normalized_set = np.log(1 + multibrot_set) / np.log(3) / np.log(maxiter)
            colored_image = (colormap(normalized_set)[:, :, :3] * 255).astype(np.uint8)
            multibrot_mobject = ImageMobject(colored_image, scale_to_resolution=2160)
            old_multibrot.become(multibrot_mobject)

        multibrot_image.add_updater(Update_multibrot)

        # >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> Tricorn <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        Tricorn_title = Tex(r"2. \quad Tricorn sets", font_size = 48).to_edge(UP)

        tricorn = multibrot_image.copy()

        def Update_tricorn(old_tricorn):
            tricorn_set = multicorn(rmin, rmax, cmax, width, height, maxiter, exp.get_value())
            normalized_set = np.log(1 + tricorn_set) / np.log(3) / np.log(maxiter)
            colored_image = (colormap(normalized_set)[:, :, :3] * 255).astype(np.uint8)
            tricorn_mobject = ImageMobject(colored_image, scale_to_resolution=2160)
            old_tricorn.become(tricorn_mobject)

        # >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> 4d Mandelbrot <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        """buhdabrot_projections = ['zr_zi',
                                 'zr_cr',
                                 'zr_ci',
                                 'zi_cr',
                                 'zi_ci',
                                 'cr_ci']

        buhdabrot_R = project_histogram_4d_to_2d(histogram = buhdabrot(rmin, rmax, cmax, width, height,
          maxiter = 100, sample_size = int(5e8)),
        projection_plane= "zi_ci",
        width = 3840,
        height = 2160,
        )

        buhdabrot_G = project_histogram_4d_to_2d(histogram = buhdabrot(rmin, rmax, cmax, width, height,
          maxiter = 500, sample_size = int(5e8)),
        projection_plane= "zi_ci",
        width = 3840,
        height = 2160,
        )

        buhdabrot_B = project_histogram_4d_to_2d(histogram = buhdabrot(rmin, rmax, cmax, width, height,
          maxiter = 1000, sample_size = int(5e8)),
        projection_plane= "zi_ci",
        width = 3840,
        height = 2160,
        )

        normalized_R = np.log(1 + buhdabrot_R) / np.log(3) / np.log(100)
        normalized_G = np.log(1 + buhdabrot_G) / np.log(3) / np.log(500)
        normalized_B = np.log(1 + buhdabrot_B) / np.log(3) / np.log(1000)

        # Combine RGB channels into one image
        combined_image = np.stack([normalized_R, normalized_G, normalized_B], axis=-1)
        combined_image = (combined_image - combined_image.min()) / (combined_image.max() - combined_image.min())
        combined_image = (combined_image * 255).astype(np.uint8)
        buhdabrot_mobject = ImageMobject(combined_image, scale_to_resolution=2160)"""

        # >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> animations <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

        with self.voiceover(text="In this scene, we will explore some fascinating generalizations of the Mandelbrot Set."):
            self.play(Write(Title), run_time=2)
        with self.voiceover(text="These include the Multibrot Sets, the Tricorn, and the 4D Mandelbrot Set."):
            self.play(Write(contents), run_time=4)
        self.wait(5)
        self.play(FadeOut(Title, contents))

        with self.voiceover(text="First, let's look at the Multibrot Sets."):
            self.play(Write(Multibrot_title))
        self.wait()
        with self.voiceover(text="The Multibrot Sets are a generalization of the Mandelbrot Set, where the exponent"
        " in the iteration formula is replaced with a diffrent degree."):
            self.play(Write(exp_indicator), FadeOut(Multibrot_title))
        with self.voiceover(text="Observe how the fractal changes as we vary the exponent."):
            self.play(exp.animate.set_value(6), run_time=15, rate_func=double_smooth)
            self.play(exp.animate.set_value(-6), run_time=20, rate_func=double_smooth)
        self.play(FadeOut(multibrot_image, exp_indicator))
        multibrot_image.clear_updaters()
        exp.set_value(0)

        with self.voiceover(text="Next, we explore the Tricorn sets."):
            self.add(tricorn)
            self.play(Write(Tricorn_title))
        self.wait()
        with self.voiceover(text="The Tricorn is generated using the iteration formula where the complex conjugate" \
        " of z is squared and added to c, instead of just z squared."):
            self.play(Write(exp_indicator), FadeOut(Tricorn_title))
        tricorn.add_updater(Update_tricorn)
        with self.voiceover(text="Let's observe the changes as we vary the exponent here as well."):
            self.play(exp.animate.set_value(6), runtime=15, rate_func=double_smooth)
            self.play(exp.animate.set_value(-6), runtime=20, rate_func=double_smooth)
        self.play(FadeOut(exp_indicator, tricorn))
```

Main.py

After that, for the next hardest scene, I made a zoom sequnce expoloring the boundry of the Mandelbrot set. This scene used the `mandelbrot_from_center` function to generate the images, and a `ValueTracker` to animate the zoom level. I used the inferno colormap to color the images, and a array of 5 points to zoom on. once agian, after wasting countless hours of work and many rounds of debugging, I was unable to meet runtime constraints. it would have taken hundreds of hours to render the scene, so I had to scrap it. **Here is the code for the scene:**

```
class ZoomSequence(VoiceoverScene):

    def construct(self):
        # Set up speech service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        # Title
        title = Text("Zooming into the Mandelbrot Set", font_size=48).to_edge(UP)
        with self.voiceover(text="Zooming into the Mandelbrot Set"):
            self.play(Write(title), run_time=2)

        # Description
        description = Text(
            "The Mandelbrot Set is infinitely complex.\n"
            "Let's explore its intricate details by zooming in.",
            font_size=36,
            line_spacing=1.5
        ).next_to(title, DOWN, buff=0.5)
        with self.voiceover(text="The Mandelbrot Set is infinitely complex. Let's explore its intricate details by zooming in."):
            self.play(Write(description), run_time=4)
        self.wait(5)

        # Zoom sequence
        zoom_points = [
            0.743643887037151 + 0.131825904205330j,
            -2 + 0j,
            -0.75 + 0j,
            0.25 + 0.5j,
            0.25+ 0j
        ]

        zoom_lvl = ValueTracker(1)

        colormap = cm.get_cmap("inferno")

        from PIL import Image
        blank_image = np.zeros((2160, 3840, 3), dtype=np.uint8)
        mandelbrot_image = ImageMobject(Image.fromarray(blank_image))

        self.add(mandelbrot_image)

        for i in zoom_points:

            def Update_zoom(old_mobject):
                mandelbrot_set = mandelbrot_from_center(centerpoint = i, zoom = zoom_lvl.get_value(),
                                                        width=3840, height = 2160, maxiter= int(zoom_lvl.get_value()) * 30)
                normalized_set = np.log(1 + mandelbrot_set) / np.log(3) / np.log(256)
                colored_image = (colormap(normalized_set)[:, :, :3] * 255).astype(np.uint8)
                mandelbrot_mobject = ImageMobject(colored_image, scale_to_resolution=2160)
                old_mobject.become(mandelbrot_mobject)

            mandelbrot_image.add_updater(Update_zoom)

            self.play(zoom_lvl.animate.set_value(500), run_time=50, rate_func=double_smooth)
            self.wait(2)
            self.play(zoom_lvl.animate.set_value(1), run_time=1, rate_func=double_smooth)
            self.wait()

            mandelbrot_image.remove_updater(Update_zoom)
```

Main.py

the final scene I made was a simple outro scene, which used a `Text` object to display the text. **The code for the scene is as follows:**

```
class outro(VoiceoverScene):
    def construct(self):
        # Set up speech service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        # Title
        title = Text("Thank you for watching!", font_size=48).to_edge(UP)
        with self.voiceover(text="Thank you for watching!"):
            self.play(Write(title), run_time=2)

        # Description
        description = MarkupText(
            "I hope you enjoyed this exploration of the Mandelbrot Set.\n"
            "If you have any questions or comments, feel free to reach out.\n"
            "email: <i><span fgcolor = 'cyan'>wenmike.xie@gmail.com</span></i>\n"
            "phone: <span fgcolor = 'red'>+1 (647) 981-4889</span>\n",
            font_size=36,
            line_spacing=1.5
        ).next_to(title, DOWN, buff=0.5)
        with self.voiceover(text="I hope you enjoyed this exploration of the Mandelbrot Set." \
        " If you have any questions or comments, feel free to reach out."):
            self.play(Write(description), run_time=4)
        self.wait(5)

        self.play(Unwrite(title), Unwrite(description))
```

Main.py

4 Code Reveal Program

although the code and process is shown here, I wanted to create a smooth way to reveal the code in the video. I wrote a new python file which breaks the code into 40-line sections, and uses a `Code` object to display the code. I really wanted people to know how much work actaully goes into 5 minutes of (crappy) animation, Which is why I made this program. Unironically the code reveal animation eneded up being longer than the actual animation. **Here is the code for the program:**

```
# type: ignore
from manim import *
from manim_voiceover import VoiceoverScene
from manim_voiceover.services.gtts import GTTSService

config.max_files_cached = 500

class code_reveal(VoiceoverScene):
    def construct(self):
        # Set up speech service
        self.set_speech_service(GTTSService(lang="en", tld="com"))

        # Title
        title = Text("Code Reveal", font_size=48).to_edge(UP)
        with self.voiceover(text="Code Reveal") as tracker:
            self.play(Write(title), run_time=tracker.duration)

        # Description
        description = Text(
            "Here is the code used to generate the Mandelbrot Set and Julia Sets.",
            font_size=30,
        ).next_to(title, DOWN, buff=0.5)

        with self.voiceover(text="Here is the code used to generate the Mandelbrot Set and Julia Sets.") as
    tracker:
            self.play(Write(description), run_time=tracker.duration)
            self.play(Unwrite(title), run_time=2)
            self.play(Unwrite(description), run_time=2)
```

```python

        self.wait(3)

        with open("Main.py", 'r') as myfile:
            chunks = []
            lines = myfile.readlines()
            chunk_size = 40
            for i in range(0, len(lines), chunk_size):
                chunks.append("".join(lines[i:i + chunk_size]))

            code_mobject = Code(
                code_string= "",
                tab_width=4,
                formatter_style="monokai",
                background="rectangle",
                language="python",
                background_config={"stroke_color": WHITE}
            ).scale(0.25)

            file_name = Text("Main.py", font_size=36).to_edge(UP, buff=0.5).shift(LEFT*3.5)
            self.play(Write(file_name), run_time=2)
            self.play(Unwrite(file_name), run_time=2)

            self.add(code_mobject)

            for ind, i in enumerate(chunks):
                self.play(code_mobject.animate.become(
                    Code(
                        code_string= i,
                        tab_width=4,
                        formatter_style="monokai",
                        line_numbers_from= ind*40 +1,
                        background="window",
                        language="python",
                        background_config={"stroke_color": WHITE}
                    ).scale(0.4)
                ), run_time=3)
                self.wait(10)

            self.play(Unwrite(code_mobject), run_time=2)


# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> part 2 >>>>>>>>>
        with open("Mandelbrot.py", 'r') as myfile:
            chunks = []
            lines = myfile.readlines()
            chunk_size = 40
            for i in range(0, len(lines), chunk_size):
                    chunks.append("".join(lines[i:i + chunk_size]))

            code_mobject_2 = Code(
                code_string= "",
                tab_width=4,
                formatter_style="monokai",
                background="rectangle",
                language="python",
                background_config={"stroke_color": WHITE}
            ).move_to(ORIGIN).scale_to_fit_height(ScreenRectangle().height + 1).scale_to_fit_width(
                 ScreenRectangle().width + 1)

            file_name = Text("Mandelbrot.py", font_size=36).to_edge(UP, buff=0.5).shift(LEFT*3.5)
            self.play(Write(file_name), run_time=2)
            self.play(Unwrite(file_name), run_time=2)

            self.add(code_mobject_2)


            for ind, i in enumerate(chunks):
                self.play(code_mobject_2.animate.become(
                    Code(
                        code_string= i,
                        tab_width=4,
                        formatter_style="monokai",
                        line_numbers_from= ind*40 +1,
                        background="window",
                        language="python",
                        background_config={"stroke_color": WHITE}
                    ).move_to(ORIGIN).scale_to_fit_height(ScreenRectangle().height + 1).scale_to_fit_width(
                         ScreenRectangle().width + 1)
                ), run_time=3)
                self.wait(10)

            self.play(Unwrite(code_mobject_2), run_time=2)

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> part 3 >>>>>>>>>>>>
```

```python
        self.play(Write(bragging := Text("Just the code reveal program itself is 190 lines",
                                          font_size = 36).to_edge(UP, buff = 0.5)), run_time=2)
        self.play(Unwrite(bragging), run_time=2)

        with open(__file__, 'r') as myfile:
            chunks = []
            lines = myfile.readlines()
            chunk_size = 40
            for i in range(0, len(lines), chunk_size):
                    chunks.append("".join(lines[i:i + chunk_size]))

            code_mobject_3 = Code(
                code_string= "",
                tab_width=4,
                formatter_style="monokai",
                background="rectangle",
                language="python",
                background_config={"stroke_color": WHITE}
            ).move_to(ORIGIN).scale_to_fit_height(ScreenRectangle().height + 1).scale_to_fit_width(
                 ScreenRectangle().width + 1)

            self.add(code_mobject_3)

            for ind, i in enumerate(chunks):
                self.play(code_mobject_3.animate.become(
                    Code(
                        code_string= i,
                        tab_width=4,
                        formatter_style="monokai",
                        line_numbers_from= ind*40 +1,
                        background="window",
                        language="python",
                        background_config={"stroke_color": WHITE}
                    ).move_to(ORIGIN).scale_to_fit_height(
                         ScreenRectangle().height + 1).scale_to_fit_width(ScreenRectangle().width + 1)
                ), run_time=3)
                self.wait(10)

            self.play(Unwrite(code_mobject_3), run_time=2)


# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> part 4 >>>>>>>>>>>>>>>

        with open(r"C:\Users\wenmi\OneDrive\Documents\GitHub\2025_ISU\GitHub-Logos\GitHub-Logos\GitHub_Logo_White.png", 'r') as logo:
            with open(r"C:\Users\wenmi\OneDrive\Documents\GitHub\2025_ISU\github-mark\github-mark-white.png", 'r') as logo_2:
                logo_mobject = ImageMobject(logo.name).scale(0.5)
                logo_mobject_2 = ImageMobject(logo_2.name).scale(0.5).next_to(logo_mobject, UP, buff=0.1)
                link = MarkupText("<i>github.com/IaMaAVerYRANdOmPerSoN/2025_ISU</i>", font_size=30
                                  ).set_color(BLUE).next_to(logo_mobject, DOWN, buff=0.1)
                link_Underline = Underline(link, buff=0.05).set_color(BLUE)

                self.play(FadeIn(logo_mobject, logo_mobject_2), run_time=2)
                self.play(Write(link),  Write(link_Underline), run_time=2)
                self.wait(3)

                self.play(FadeOut(logo_mobject, logo_mobject_2), Unwrite(link), Unwrite(link_Underline), run_time=2)

        self.play(Write(creds := Text("Made with:", font_size=48).to_edge(UP, buff=0.5)))
        self.play(Unwrite(creds))

        banner = ManimBanner()
        self.play(banner.create())
        self.play(banner.expand())
        self.wait(2)
        self.play(Unwrite(banner))

        numpy_logo = ImageMobject(r"C:\Users\wenmi\OneDrive\Documents\GitHub\2025_ISU\numpylogo.png")

        self.play(FadeIn(numpy_logo), run_time=2)
        self.play(FadeOut(numpy_logo), run_time=2)

        matplotlib_logo = ImageMobject(r"C:\Users\wenmi\OneDrive\Documents\GitHub\2025_ISU\sphx_glr_logos2_003_2_00x.webp")

        self.play(FadeIn(matplotlib_logo), run_time=2)
        self.play(FadeOut(matplotlib_logo), run_time=2)

        vscode_logo = ImageMobject(r"C:\Users\wenmi\OneDrive\Documents\GitHub\2025_ISU\Visual_Studio_Code_1.35_icon.svg.png").scale(0.5)

        self.play(FadeIn(vscode_logo), run_time=2)
        self.play(FadeOut(vscode_logo), run_time=2)
```

5 Challenges and debugging

I encountered numerous challenges while working on this project. The most difficult aspect was debugging the code and ensuring that the animations rendered properly. I spent countless hours trying to figure out why certain sections weren't functioning as intended, often rewriting large portions of the code multiple times. Learning how to use Manim and its features was also a steep learning curve. The documentation was oftentimes unclear, so I had to rely heavily on online forums and community support. People often underestimate how difficult coding can be and how much time it takes to get things working correctly. Imagine if Google Docs took ten minutes to load every time you wanted to edit a document, and if you made a mistake, you'd have to wait another ten minutes just to see if it worked. And instead of a user-friendly interface, you had to write everything in a plain text file. That's what coding can feel like: time-consuming, frustrating, and mentally exhausting.

Sometimes it gets so bad that *unironically* 40% of your video doesn't even exist because it was bugged—something I experienced firsthand. I had to turn to Stack Overflow multiple times, once waiting a week just for a reply. Unlike a human, the Python interpreter doesn't understand your intentions; it only interprets the code you wrote. A single typo or subtle logic error can break everything, and the error messages are often vague—or it might even fail silently. To make things more challenging, my subject matter involved complex mathematics and abstract thinking, which much added more room for error. And as if that weren't enough, imagine that if your English were grammatically correct but not perfectly clear and concise, Google Docs would take 1,000 times longer to load and you'd need to learn even more math just to speed things up. That's what programming felt like throughout this project.