Generative Panning with Outpainting

Kayla Akyüz
kaylaakyuz@gmail.com
kaylaakyuz@hacettepe.edu.tr
Hacettepe University

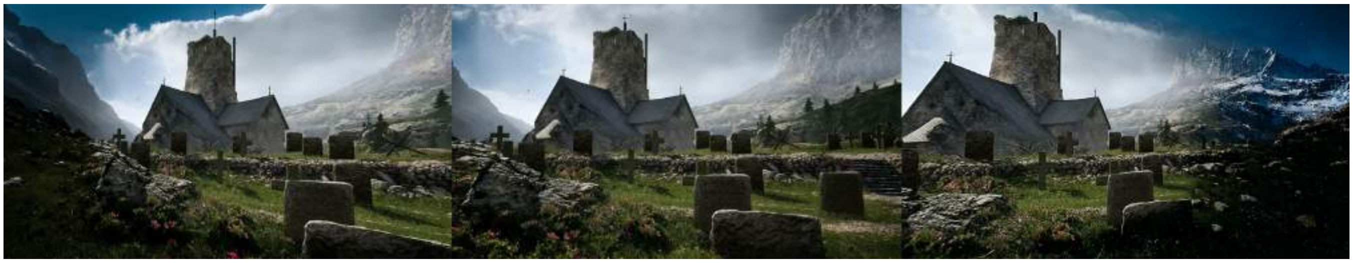


Figure 1: **A Generated In Game Image**

ABSTRACT

As a term project for BBM444 course at Hacettepe University, I have explored and researched the concept of generative panning with the usage of outpainting. In this document you will find my final report which includes references to the code, results and user study conducted. I will be explaining my method and ideas, discussing obtained results and comparing them with related works. At the end I will be giving insight in to the future of this research, leading towards simulated virtual realms.

KEYWORDS

outpainting, neural networks, stable diffusion, camera panning

1 INTRODUCTION

The core idea is to investigate and develop solutions for completely AI-generated games, focusing on image-to-image generation. This concept requires extensive research into several aspects, beginning with the ability to generate 3D environments, maintain context while looking around, and retain continuity. Once these fundamentals are established, we can further explore user interactivity and gamification elements.

The ultimate goal is to create generated realms that allow users to craft virtual environments via multi-modal AIs. There are numerous approaches to achieve this, the simplest being the conversion of a given base 2D image into 360° panoramic images. However, this approach seems less feasible as panoramic images lack sufficient research for image modification and are primarily used for looking around rather than for creating interactive games.

A deeper reasoning I envisioned involves generating realms image by image, giving the user the illusion of free will. Consider how the main side effect of panning with outpainting resembles human dreams, where turning around might reveal a changed scene or missing objects. This occurs because our brain doesn't simulate the entire world at once; instead, it provides flashes of images that create the sense of a 3D world. Similarly, when we are awake, we perceive the 3D world, but we don't simulate the entire 3D environment. Our eyes only see a 2D slice, and our understanding of the 3D world changes based on the direction we face. If this weren't the case, humans could perfectly sense whatever is outside their field of view.

What does this entail for simulated worlds or games? Imagine a user looking at a desk, then reading a note that says to turn around and face a mystery. In a 360° generated game, whatever is behind the user is already generated before they turn around. However, in the panning with outpainting scenario, the user's journey while turning around influences the end result. Although this is deterministic, it gives the impression of user interaction affecting the outcome, which feels like free will.

Thinking in these terms reveals that to reach simulated realms, we should focus on generating a concise and contextually appropriate series of 2D images instead of 360° images or whole 3D models. This approach better simulates the experience of free will and interaction within a dynamically generated world.

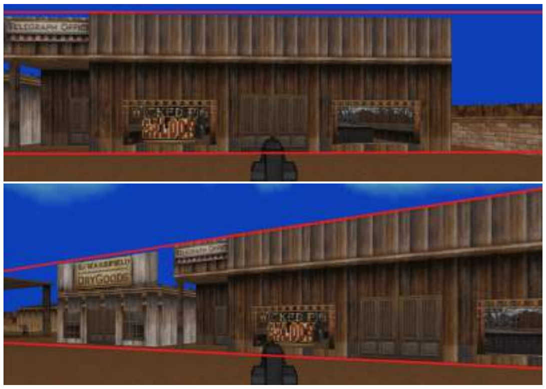


Figure 2: **We can observe the effect of panning in the games.**

After understanding this aspect, we can proceed with expanding in this area. I have taken steps to address one of the initial challenges: panning. I have devised a method that poses the question, **"Is it possible to pan the camera with outpainting on warped images, and what are the caveats"**. This research and the developed solutions have yielded feasible results, and I will report on the steps taken to reach this outcome.

My devised solution came to me in a moment of contemplation. It stemmed from a long-standing perspective I had been pondering about creating games solely from image generation. The solution is simple: warping images in a way to simulate panning of the camera, then outpainting the remaining region. Initially, I questioned whether this approach was even feasible. Would AI be able to capture the rotation, resulting in a convincingly panned image, or would it appear distorted and glitched? As you might already know if you have peeked at the results, AI does capture the panning effect, provided certain methods are employed to achieve it perfectly. However, there are deeper questions and nuances that require extensive fine-tuning within the concept. For instance, panning might alter occlusions within the field of view (FOV) and perspective of the cameras, revealing new objects. If we simply warp and outpaint, we might overlook these specific details. Another crucial consideration is that objects further from the camera will move slower, while those closer will move faster. In other words, the chosen image must possess a certain depth, and its perspective and FOV must fall within a certain range to achieve the optimal result. Otherwise, we need to assess the depth of the image and adjust our functions accordingly to accommodate its perspective and depth.

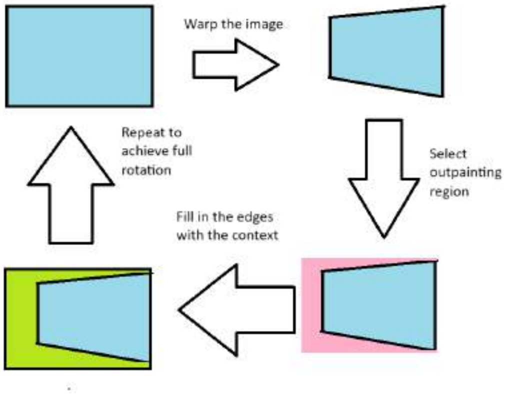

Warp the image
Select outpainting region
Fill in the edges with the context
Repeat to achieve full rotation

Figure 3: **The initial idea which evolved as research progressed, transitioning to a more complex concept.**

This whole intricacy can be bypassed by simply switching to a trained neural network to carry out initial panning. We can create a dataset by placing a camera in the world and another one very close, but panned. By adjusting final images, we can obtain a dataset, or we can even use in-game computer graphics to populate a dataset. Then, we can train the resulting AI, which would hopefully capture all of the mentioned and unforeseen details of panning.

While this might be a consideration for the future, let's dive into my solution, which aims to provide inspiration and produce a tool that generates moderately appealing (based on the user study) panning results. It serves as a foundation for further expansion and a banner to raise towards creating 3D realms through 2D image-to-image generation.

2 RELATED WORK

I will discuss the similar methods that were mentioned in the previous documents. However, this time, I also have a user study result comparing Adobe's Beyond the Seen with my results. I obtained their exact photo and generated panning, and added it as a section in the user study conducted.

To kick things off, this work couldn't have been conducted without the numerous related works addressing the broader question of this idea, including specialized works focusing on camera movements and panning. Additionally, tools and technologies have been developed to facilitate this development. As they say, "If you wish to make an apple pie from scratch, you must first invent the universe." [13]

The research aimed at finding similar works revealed that the exact idea of warping and outpainting images is original and unique. While there is another method, which involves generating 360° images and then panning as desired, I have discussed this in the introduction and will delve further into it in the results section. However, for now, let's move on to the related projects that I mentioned before.

Firstly, we can examine the Corridor Crawler Outpainting [1], which is essentially testing out a very specific context of camera movement: zooming out in corridors. The results appear satisfactory, and when played in reverse, they create the illusion of a corridor, reminiscent of early DOS-era games.

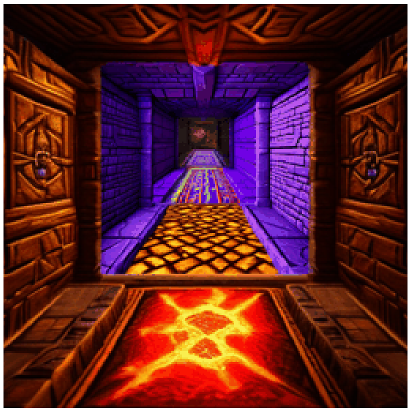


Figure 4: **Corridor Crawler Outpainting.**

Another related work comes from the renowned Midjourney, where they delve into the concept of panning. [2]

Figure 5: **Panning banner generated with Midjourney.**

Their approach to panning involves extending the canvas in the given direction without any morphing to the original image. While this differs from my envisioned camera panning, it still seems to capture the movement of the camera and enables users to immerse themselves in a further envisioned world where the borders are expanded in either direction.

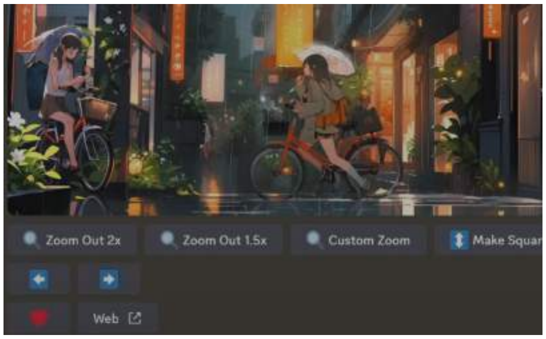


Figure 6: **Panning interface of Midjourney. [3]**

Some users even appear to be able to look around if the given image is already perspective-warped; however, further expansion generates extremely warped edges. The question arises: if implemented differently, by regenerating the entire image to include a panning effect, can we achieve true rotation or even a complete 360-degree rotation?

Another work that caught my attention is the 360-drawing.com's 360 painting tool, which seems to offer the ability to paint 360 images with generative painting. Their video on the website provides a detailed showcase of the tools, demonstrating their capability to outpaint in any given direction of the x, y, z plane of a 3D, 360 image.

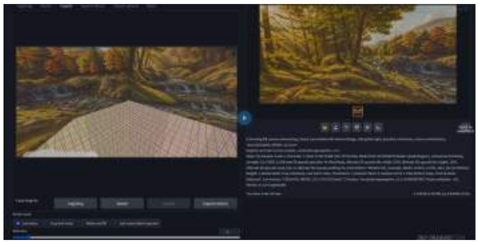


Figure 7: **360-drawing.com's tool that generates images in a 3D plane.**

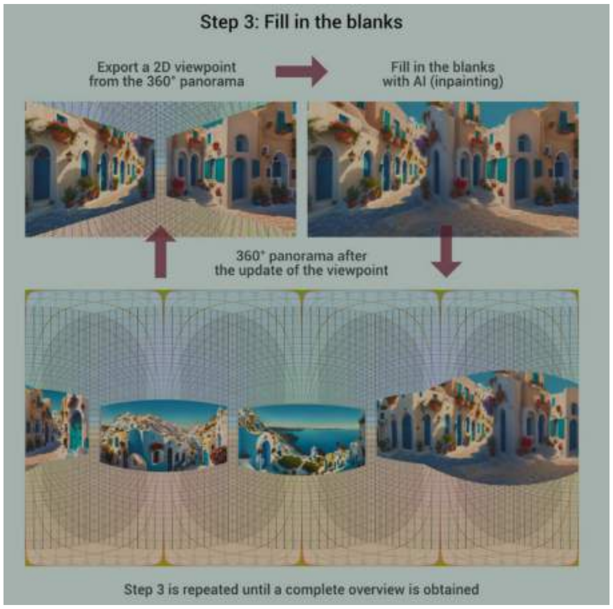

Step 3: Fill in the blanks
Export a 2D viewpoint from the 360° panorama
Fill in the blanks with AI (inpainting)
360° panorama after the update of the viewpoint
Step 3 is repeated until a complete overview is obtained

Figure 8: **Kevin Hohler's 360 panorama image generation. [4]**

This idea is explored in a couple of other works, such as the article "How to create 360 Panoramas using Artificial Intelligence" [4]. In this article, Kevin Hohler explains a similar method of visualizing a 3D environment by generating the image in a 360 panorama format.

Another work that expands on the same method is "Beyond The Seen" by Adobe, where they pose the question: "Have you ever wanted to create immersive 360° experiences from flat 2D images?" [5]

Figure 9: **"Beyond The Seen" by Adobe**

Additionally, I must acknowledge a couple of outpainting tools that have paved the way for these developments, such as Stable Diffusion itself, as well as Fooocus [8] for their user interface, and the Stability Matrix [9] for their package management tool for stable diffusion.

3 THE APPROACH

My original idea was simply to warp images and outpaint. However, as the research progressed, it became evident that there's more to warping than simply manipulating the edges and corners. The warping process is a complex flow that needs to be depth-guided. Since I couldn't train such a model, I implemented a simpler solution by dividing the image into four segments. Each segment is then warped differently according to the direction.

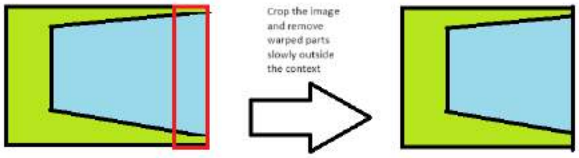

Crop the image and remove warped parts slowly outside the context

Figure 10: **A fix via cropping.**

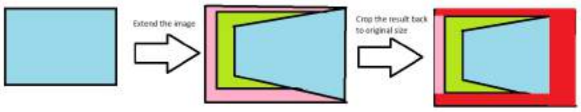


Figure 11: **A more intricate methodology was employed, reflecting modifications to the initial idea.**

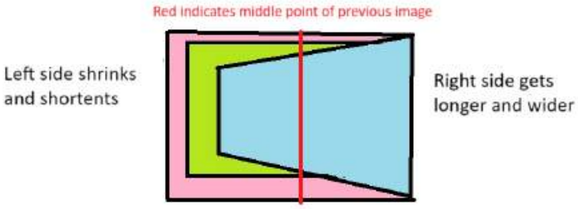

Red indicates middle point of previous image
Left side shrinks and shortents
Right side gets longer and wider

Figure 12: **Additional caveats to get realistic panning.**

I utilized the OpenCV library [10] for the warpPerspective and findHomography functions. The complete code is shared via a Colab notebook at this Colab Notebook[14]

For outpainting, I initially utilized InvokeAI [11], but later transitioned to Fooocus [8] due to its simpler interface. Fooocus refers to the process as inpainting when dealing with defined regions, whereas outpainting is designated for undefined areas. On the other hand, InvokeAI's multi-layer canvas automatically identifies the operation as outpainting. Ease to get higher quality generations, and importance of an API made me opt in for Fooocus.

```python
import cv2
import numpy as np
import requests
import json
import numpy as np
from PIL import Image
from io import BytesIO

host = "http://127.0.0.1:8888" # Adress of the Fooocus API
model = "juggernautXL_juggernautX.safetensors" # Model for generation

def warp_image(image, angle_degrees, direction):
    # Convert angle from degrees to radians
    angle_radians = np.radiaans(angle_degrees)

    # Get image dimensions
    height, width = image.shape[:2]

    offset = 0
    mask_padding = 15
    mask_destionation = np.float32([
            [0, 0],
            [0, 0],
            [0, 0],
            [0, 0],
            [0, 0],
            [0, 0],
            [0, 0],
            [0, 0]
        ])

    if direction == 'up' or direction == 'down':
        offset = int(width * np.tan(angle_radians) * 0.5)
    elif direction == 'left' or direction == 'right':
        offset = int(height * np.tan(angle_radians) * 0.5)

    height = height + (offset*2)
    width = width + (offset*2)
    image = cv2.copyMakeBorder(image, offset, offset, offset, offset, cv2.BORDER_CONSTANT, value=[0, 0, 0])

    # Define 8 points for perspective transformation
    src_points = np.float32([
        [offset, offset],                 # Top-left corner
       [(width - 1) // 2, offset],        # Top-middle
        [width - 1 - offset, offset],         # Top-right corner
        [width - 1 - offset, height // 2],# Right-middle
        [width - 1 - offset, height - 1 - offset], # Bottom-right corner
        [(width - 1) // 2, height - 1 - offset],# Bottom-middle
        [offset, height - 1  - offset],        # Bottom-left corner
        [offset, height // 2]        # Left-middle
    ])

    if direction == 'up':
        dst_points = np.float32([
            [offset*2, offset*2],
            [(width - 1 )//2, offset*2],
            [(width - 1 ) - (offset*2), offset*2],
            [width - 1 - offset, (height -1)//2],
            [width - 1, (height -1)],
            [(width - 1 )//2, (height -1)],
            [0, height - 1],
            [offset, (height -1)//2]
        ])
        mask_destionation = np.float32([
            [offset*2 + mask_padding, offset*2 + mask_padding],
            [(width - 1) - (offset*2) - mask_padding, offset*2 + mask_padding],
            [(width - 1) - (mask_padding*2), (height -1)],
            [(mask_padding*2), (height -1)]
        ])

    elif direction == 'down':
        dst_points = np.float32([
            [0, 0],
            [(width - 1 )//2, 0],
            [width - 1, 0],
            [width - 1 - offset, (height -1)//2],
            [width - 1 - (offset*2), (height - 1) - (offset*2)],
            [(width - 1 )//2, (height - 1) - (offset*2)],
            [offset*2, (height - 1) - (offset*2)],
            [offset, (height -1)//2]
        ])
        mask_destionation = np.float32([
            [mask_padding*2, 0],
            [width - 1 - (mask_padding*2), 0],
            [width - 1 - (offset*2) - (mask_padding), (height - 1) - (offset*2) - (mask_padding)],
            [(offset*2) + mask_padding, (height - 1) - (offset*2) - mask_padding]
        ])

    elif direction == 'left':
        dst_points = np.float32([
            [offset*2, offset*2],
            [(width - 1 )//2, offset],
            [width - 1 , 0],
            [width - 1 , (height -1)//2],
            [width - 1 , height - 1],
            [(width - 1 )//2, (height -1) - (offset)],
            [offset*2, (height -1)-(offset*2)],
            [offset*2, (height -1)//2]
        ])
        mask_destionation = np.float32([
            [offset*2 + mask_padding, offset*2 + mask_padding],
            [width - 1, mask_padding*2],
           [width - 1, (height -1) - (mask_padding*2)],
            [offset*2 + mask_padding, (height -1)- (offset*2) - mask_padding]
        ])
    elif direction == 'right':
        dst_points = np.float32([
            [0, 0],
            [(width - 1 )//2, offset],
            [(width - 1) - (offset*2) , offset*2],
           [(width - 1) - (offset*2) , (height -1)//2],
            [(width - 1) - (offset*2) , (height - 1) - offset*2],
            [(width - 1 )//2, (height -1) - offset],
            [0, height - 1],
            [0, (height -1)//2]
        ])
        mask_destionation = np.float32([
            [0, mask_padding*2],
            [(width - 1) - (offset*2) - mask_padding, offset*2 + mask_padding],
            [(width - 1) - (offset*2) - mask_padding, (height - 1) - (offset*2) - mask_padding],
            [0, (height -1) - (mask_padding*2)],
        ])


    else:
        raise ValueError("Invalid direction. Use 'left', 'right', 'up', or 'down'.")

    # Calculate the perspective transformation matrix
    matrix, _ = cv2.findHomography(src_points, dst_points)

    # Apply the perspective transformation
    warped_image = cv2.warpPerspective(image, matrix, (width, height), flags=cv2.INTER_CUBIC)

    # Create a mask image
    mask_image = np.zeros_like(warped_image)
    mask_image = cv2.fillPoly(mask_image, [np.int32([[0, 0], [warped_image.shape[1], 0], [warped_image.shape[1], warped_image.shape[0]], [0, warped_image.shape[0]]])], (255, 255, 255))

    # Calculate the mask perspective transformation matrix
    mask_image = cv2.fillPoly(mask_image, [np.int32(mask_destionation)], (0, 0, 0))

    return warped_image, mask_image, offset

def convert_image_to_byte(image):
  image_pil = Image.fromarray(cv2.cvtColor(image,
 ↪  cv2.COLOR_BGR2RGB))
  bytes_io = BytesIO()
  image_pil.save(bytes_io, format='PNG')
  return bytes_io.getvalue()

def inpaint_outpaint(params: dict, input_image: bytes,
↪  input_mask: bytes = None) -> dict:
    """
    example for inpaint outpaint v1
    """
    response =
    ↪  requests.post(url=f"{host}/v1/generation/image-inpai
                        data=params,
                        files={"input_image":
                        ↪  input_image,
                            "input_mask": input_mask})
    return response.json()

def run_warp_pipeline(image, degree, direction,
↪  save=False, display=False):
    warped_image, mask_image, offset =
    ↪  warp_image(image, degree, direction)
    if save:
        cv2.imwrite('warped_image.png', warped_image,
        ↪  [cv2.IMWRITE_PNG_COMPRESSION, 0])
        cv2.imwrite('mask_image.png', mask_image,
        ↪  [cv2.IMWRITE_PNG_COMPRESSION, 0])
    if display:
        cv2.imshow('Warped Image', warped_image)
        cv2.imshow('Mask Image', mask_image)
        cv2.waitKey(0)
        cv2.destroyAllWindows()

    return convert_image_to_byte(warped_image),
    ↪  convert_image_to_byte(mask_image), offset

def run_outpaint_pipeline(source, mask, prompt="",
↪  async_process=True):
  result = inpaint_outpaint(params={
                            "prompt": prompt,
                            "negative_prompt": "dark,
                            ↪  shadow, dark shadows,
                            ↪  columns, obstructions,
                            ↪  blocked view, close
                            ↪  object, close-up,
                            ↪  frame, obstacles,
                            ↪  nearby objects,
                            ↪  rear-view mirror,
                            ↪  looking out car,
                            ↪  looking out of a car,
                            ↪  logo, banner, UI, HUD,
                            ↪  GUI",
                            "async_process":
                            ↪  async_process,
                            "base_model_name": model
                            },
                          input_image=source,
                          input_mask=mask)
  #
  return result

def continuous_pipeline(image, direction, degree,
↪  times, prompt):
    for _ in range(times):
        warped_image, mask_image, offset =
        ↪  run_warp_pipeline(image, degree,
        ↪  direction, save=False)
        result = run_outpaint_pipeline(warped_image,
        ↪  mask_image, prompt=prompt,
        ↪  async_process=False)
       # Fetch the image from the URL and read it into
        ↪  OpenCV format
        response = requests.get(result[0]['url'])
        image_data = BytesIO(response.content)
        image_array =
        ↪  np.asarray(bytearray(image_data.read()),
        ↪  dtype=np.uint8)
        image = cv2.imdecode(image_array,
        ↪  cv2.IMREAD_COLOR)

        # crop from the opposite direction as much as
        ↪  the offset
        if direction == 'up':
            image = image[:-offset*2, offset:-offset,
            ↪  :]
        elif direction == 'down':
           image = image[offset*2:, offset:-offset, :]
        elif direction == 'left':
            image = image[offset:-offset, :-offset*2,
            ↪  :]
        elif direction == 'right':
           image = image[offset:-offset, offset*2:, :]
    return image

def look_around(image, degree_increment, path, save =
↪  False, step_increment = 1, prompt = ""):
  img = image
  image_no = 0
  for direction in path:
    img = continuous_pipeline(img, direction,
    ↪  degree_increment, step_increment, prompt)
    if save:
      cv2.imwrite(str(image_no) + '_' + direction +
      ↪  '.png', img, [cv2.IMWRITE_PNG_COMPRESSION,
      ↪  0])
    image_no += 1

path_image = cv2.imread('game.png')

path = ['right'] * 35
path_angle = 5
look_around(path_image, path_angle, path)
```

At the end, it's as simple as defining a path and providing the image. The user can add additional prompts if they desire. I suggest viewing the document in Google Colab as well.

As you can see in the code, first, we obtain 8 points relative to the starting point from the top right corner and move clockwise along the edges. Then, depending on our direction, we warp these points as described in Figure 11 and 12. We obtain the resulting points using the findHomography function, and we also extract the mask, which will guide outpainting. Finally, we use the warpPerspective function to warp the image.

Outpainting is performed simply by sending the image, mask and prompts to the Fooocus API.

There are several functions aimed at streamlining the process and condensing the code. For instance, we have the look_around pipeline, which applies the function iteratively as necessary to achieve larger panning degrees through smaller increments. The resulting image is then stored in the outputs folder of the API, or we can utilize the built-in saving mechanism to save the result in any desired folder.

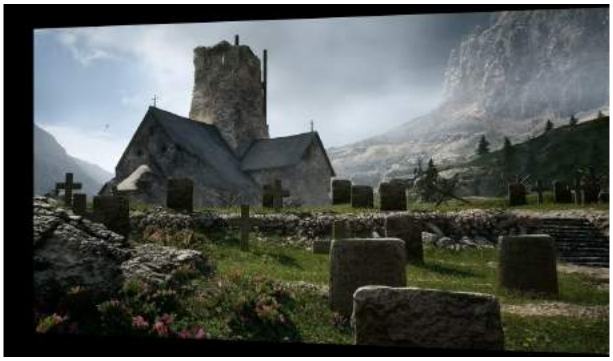


Figure 13: **An example warped image.**

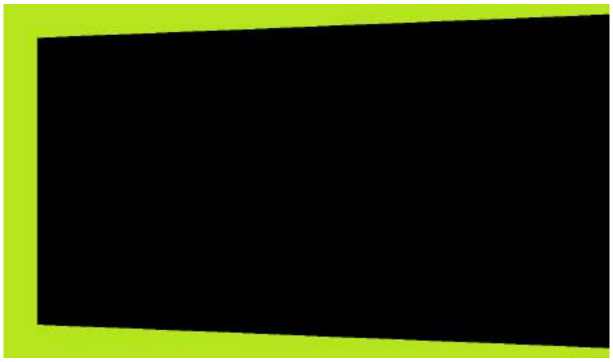


Figure 14: **An example mask.**

4 EXPERIMENTAL RESULTS

The initial results were not as promising, however there were a milestone to establish the pipeline.

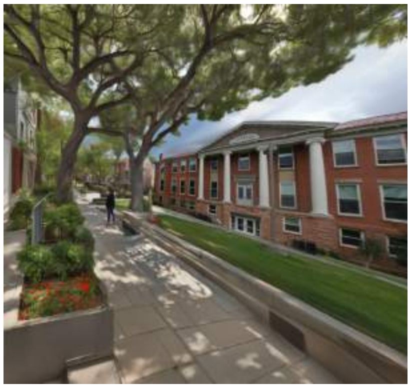


Figure 15: **One of the initial results.**

After switching to Fooocus and Juggernaut X model, adjusting negative prompts and warping with a proper pipeline through the OpenCV, I got way better results.

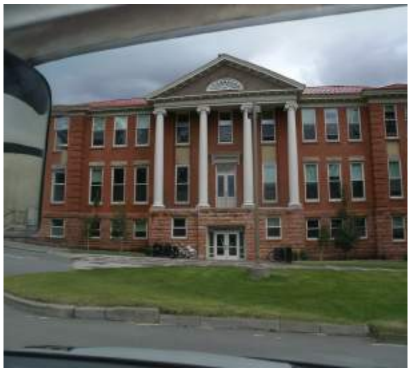


Figure 16: **AI generates the perspective of looking out of a car.**

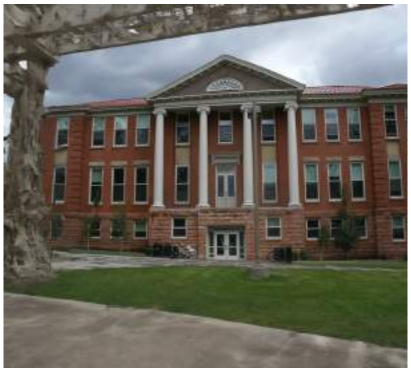


Figure 17: **AI generates arches**

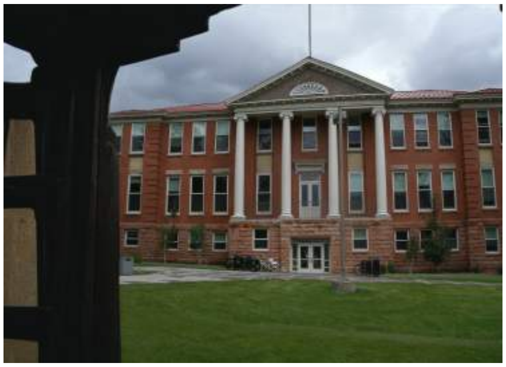


Figure 18: **AI generates frames.**

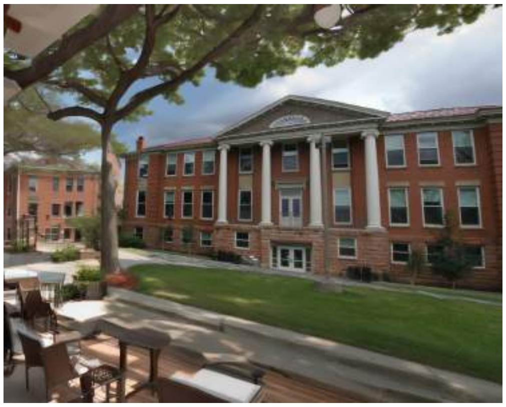


Figure 19: **Considering the whole image will eliminate out-of-context objects.**

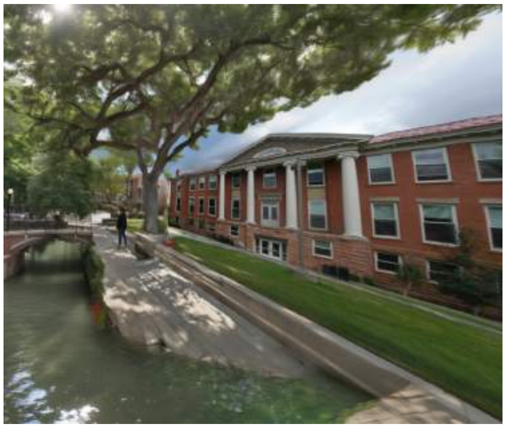


Figure 20: **The outpainting region must be chosen precisely. Otherwise giving too much space will result AI to generate glitching results.**

Even after switching to the new pipeline, there were many failed results. However, after addressing each caveat, I continued moving closer towards the goal.

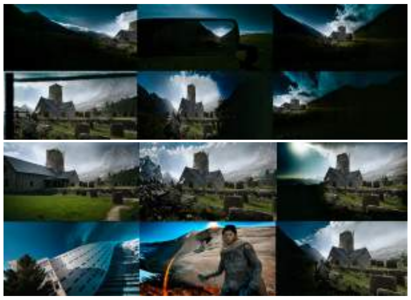


Figure 21: **There were thousands of failed results.**

Now let's delve into some of the actual results. However, I'm only able to add images to the PDF document, whereas the results are better viewed in video format. You can find all of the result videos in the shared folder via "Mega Link" [15]

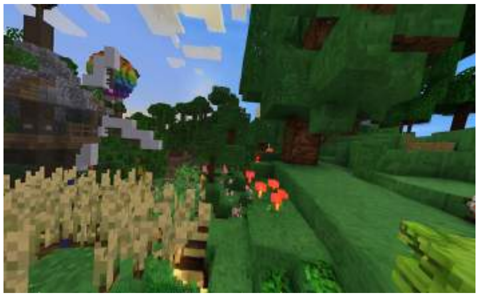


Figure 22: **AI generated panning in Minecraft.**

To compare with ground truth results, I captured in-game panning videos and then generated similar videos using the panning method on the first frame of the screen capture. These videos were part of the user study and were compared with other methods. The ground truth videos can be found in the shared Mega folder.

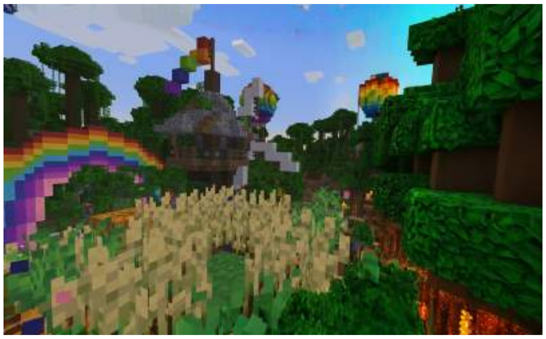


Figure 23: **Notice the balloon on the right, added by the AI, capturing the essence of the balloons prior to panning on the left.**

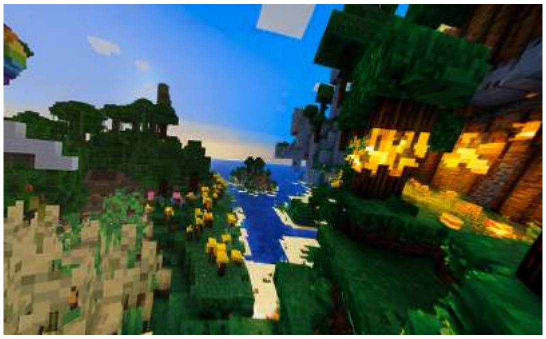


Figure 24: **Despite the high FOV, the AI successfully achieves panning.**

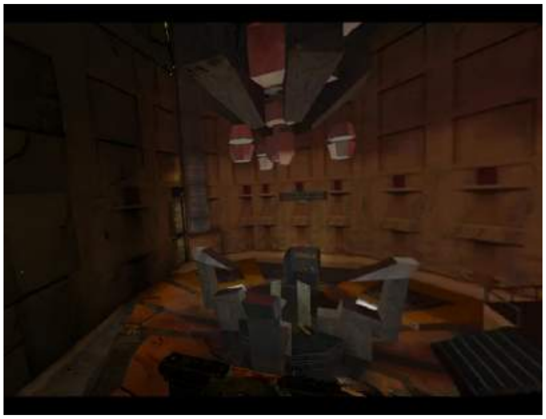


Figure 25: **The famous scene from Half-Life panned to the left by AI. Almost looks same as the game, with only the wall differing.**

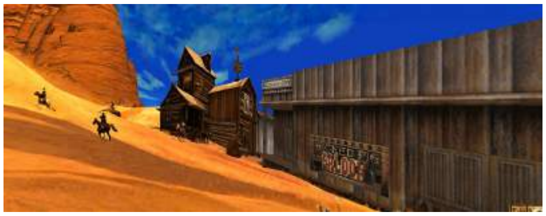


Figure 26: **A legendary game, Outlaws (1997), which was mentioned before; AI captures the essence of the scene perfectly.**

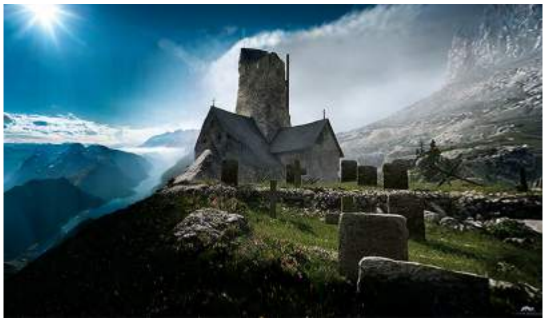


Figure 27: **A warped hillside from an in-game screenshot, demonstrating impressive results. Watch the full video on Mega for more details.**

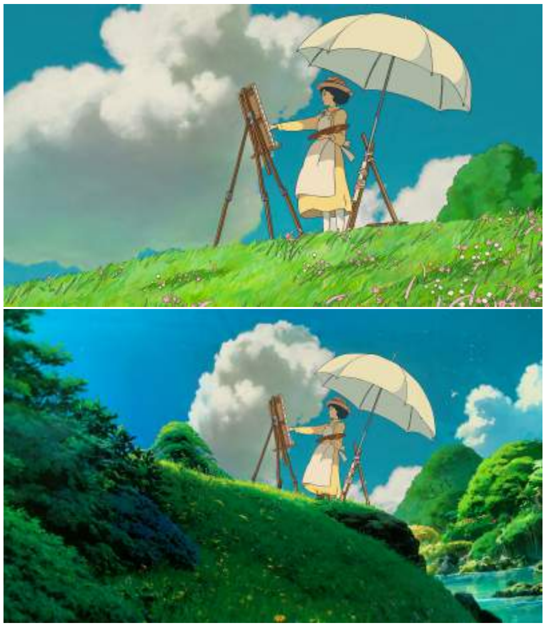


Figure 28: **An impressive panning effect in the anime style.**

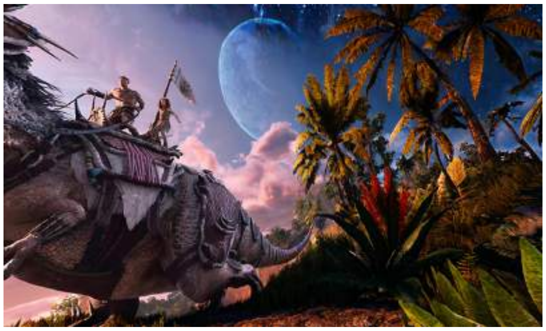


Figure 29: **Even a screenshot from the famous game Ark is generated in context, appearing as if straight from the game.**

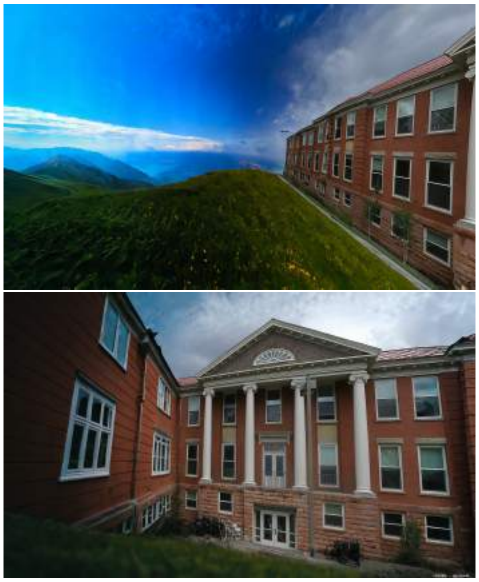


Figure 30: **Remember the initial input image; now we can generate fields or buildings that look realistic.**

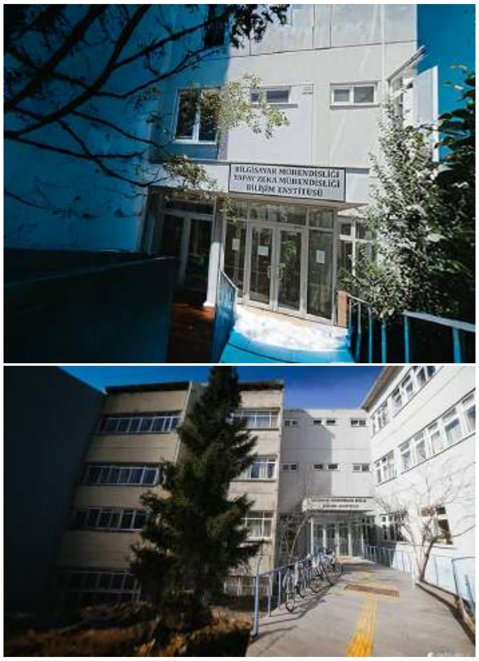


Figure 31: **Our own school building, Hacettepe University's Computer Engineering Department. Must see results at the Mega link.**

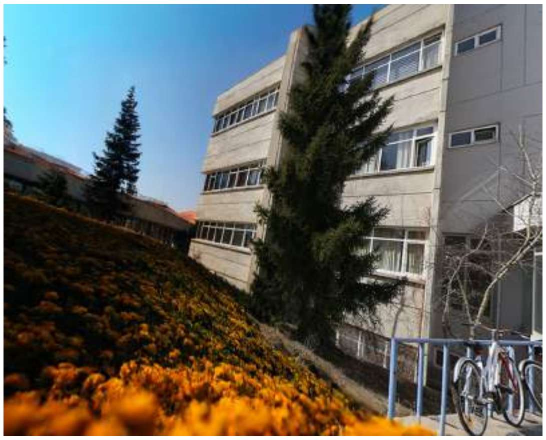


Figure 32: **Especially this one where panning was done 360 degrees. A must-watch to understand the results.**

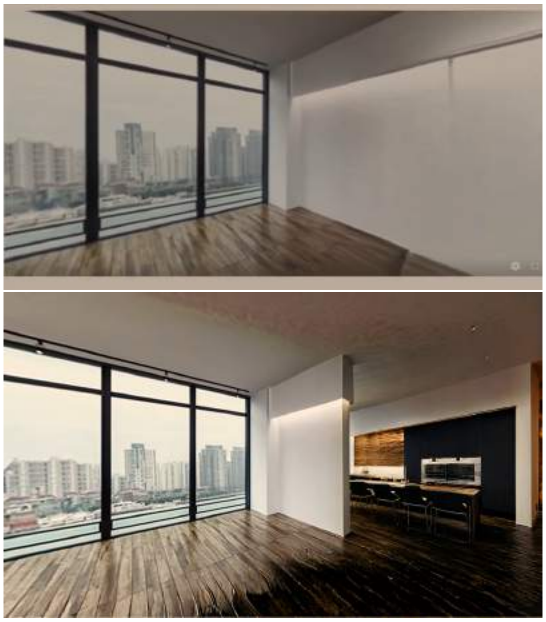


Figure 33: **Adobe's result displayed above, my result shown below, illustrating variety and quality.**

After analyzing all these results and comparing them with ground truth images, I further conducted tests using the exact images for Adobe Beyond the Seen presentation. [5].

These results were also affirmed in the conducted user study. You can access it via this link [16].

The user study was shared among colleagues, teachers, and assistants. The main participation came from other universities, as I shared the link in many WhatsApp groups. Initially, the target audience was limited to imminent students who take the same class. As sufficient participation was not achieved, I gradually expanded the target audience to include students from other universities and departments. The results are cut off at number 27 in the submission, even though the form is still open to be viewed.

You can find the end results in the previously shared link [15]. You can also view the sheet on Google Docs [17].

Also I have created charts and report analysis at Google Lookers Studio. [18] In the Lookers Studio, you can explore the charts interactively. Simply click on a slice to select a filtering in a chart, which will be applied to all others.

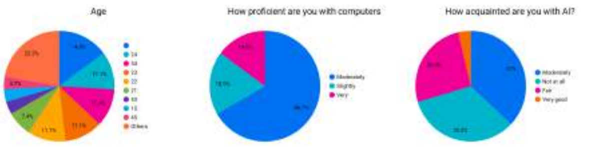


Figure 34: **You can select from demographics or any other slice to filter results.**

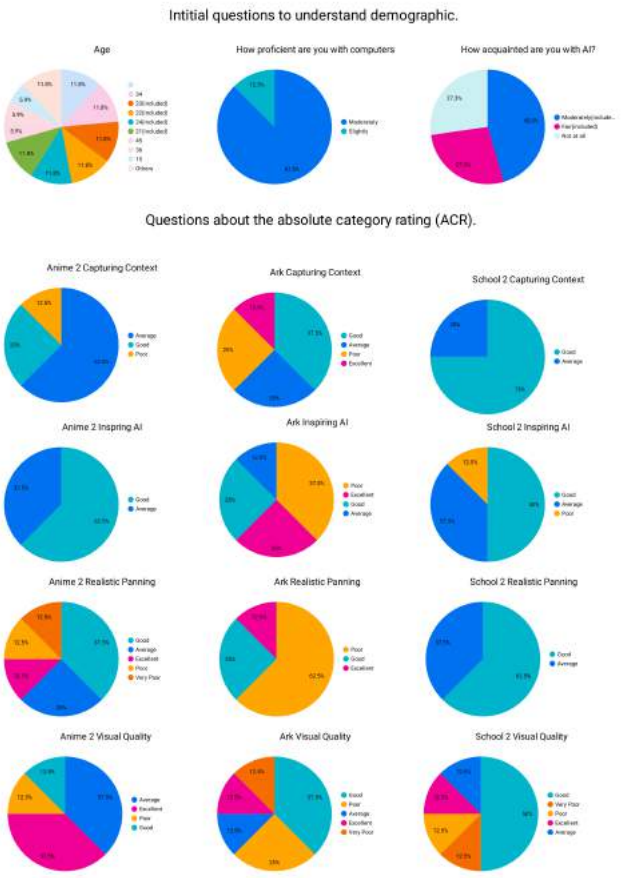


Figure 35: **You can hold Ctrl and select multiple filters from multiple charts. In this figure, I have chosen individuals in their twenties and those who are familiar with AI.**

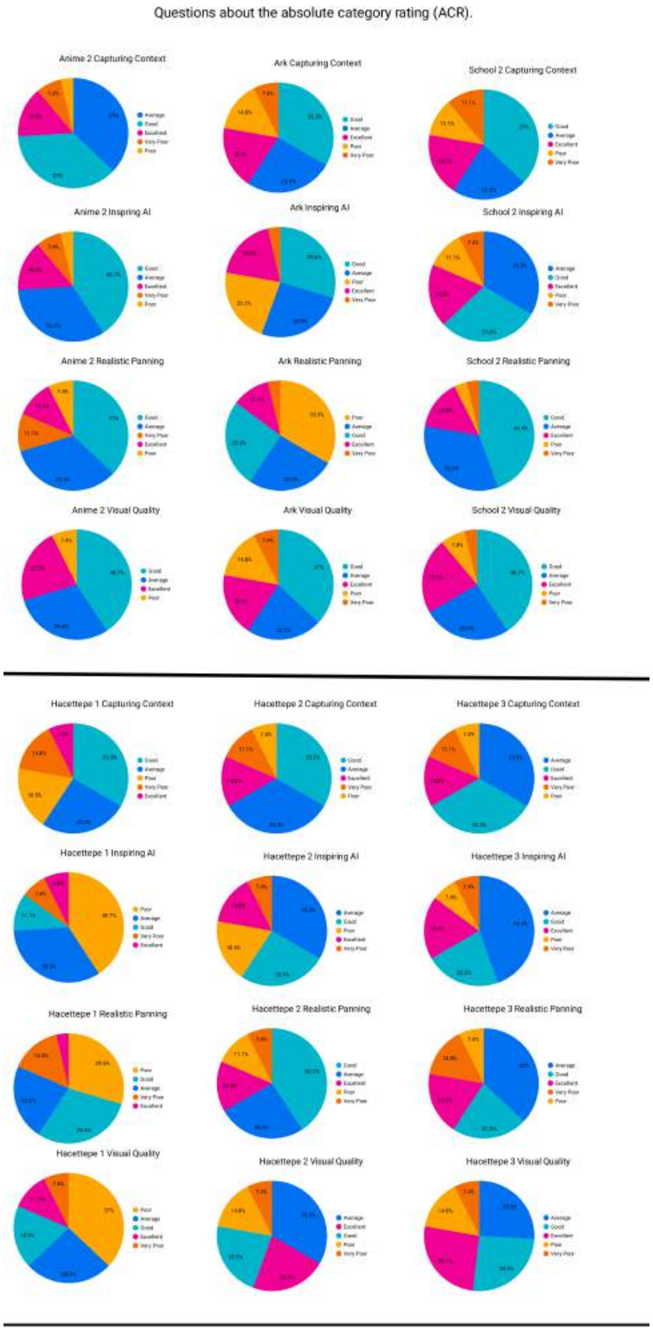
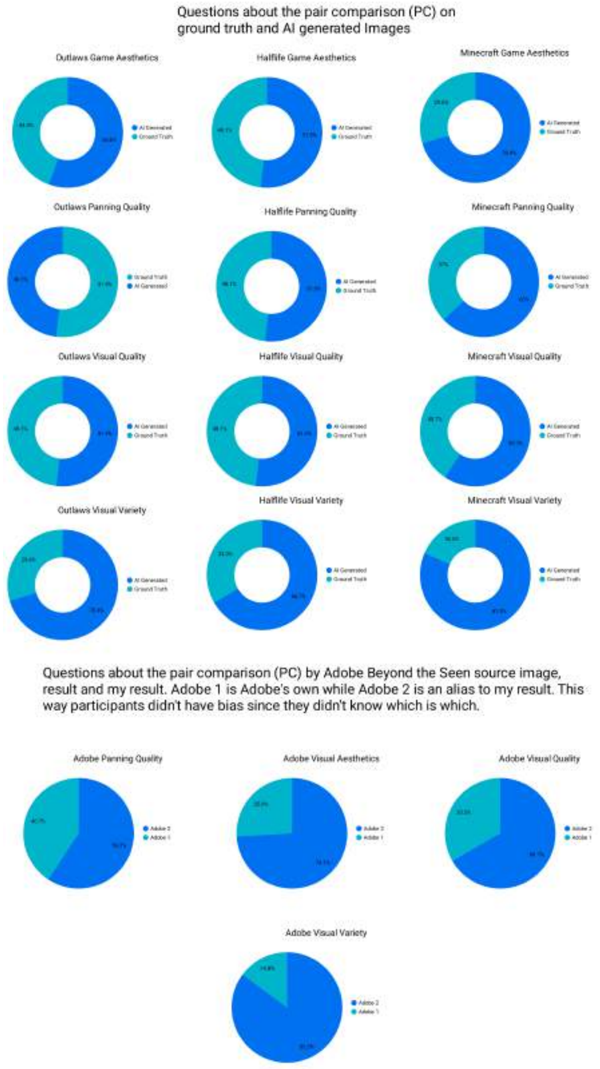


Figure 36: **Unfiltered results of absolute category ratings. Visit the provided link for interactive filtering.**

Participants in the study rated multiple attributes of different AI-generated panning videos and compared them with ground truth videos. The attributes rated included; Visual Quality, Realistic Panning, Inspiring AI, Capturing Context.



Questions about the pair comparison (PC) on ground truth and AI generated Images

Questions about the pair comparison (PC) by Adobe Beyond the Seen source image, result and my result. Adobe 1 is Adobe's own while Adobe 2 is an alias to my result. This way participants didn't have bias since they didn't know which is which.

Figure 37: **Pair comparison charts.**

Participants were also asked to compare pairs of videos on attributes like visual quality, visual variety, game aesthetic, and panning quality. Finally, participants provided their opinions on whether the research could lead to the creation of games with AI-generated panning.

The data collected includes responses from participants of varying ages and levels of proficiency with computers and AI. The responses cover ratings for multiple AI-generated videos and ground truth comparisons. Participant Demographics:

- Age Range: 15 to 52 years
- Proficiency with Computers: Slightly to Very proficient
- Acquaintance with AI: Not at all to Very acquainted

Participants were asked to select which video in a pair had better attributes. The pairs compared were in game panning from Outlaws (1997), Half-life, Minecraft, and the panning of exact image used for Adobe Beyond the Seen.

- Outlaws
  - Visual Quality: Majority preferred AI Generated
  - Visual Variety: Majority preferred AI Generated
  - Game Aesthetic: Mixed preferences
  - Panning Quality: Majority preferred AI Generated
- Halflife
  - Visual Quality: Mixed preferences
  - Visual Variety: Mixed preferences
  - Game Aesthetic: Mixed preferences
  - Panning Quality: Majority preferred Ground Truth
- Minecraft
  - Visual Quality: Mixed preferences
  - Visual Variety: Mixed preferences
  - Game Aesthetic: Mixed preferences
  - Panning Quality: Majority preferred Ground Truth
- Adobe
  - Visual Quality: Majority preferred Adobe 2
  - Visual Variety: Majority preferred Adobe 2
  - Visual Aesthetic: Majority preferred Adobe 2
  - Panning Quality: Majority preferred Adobe 2

For the game ground truths, only the Outlaws generation passed, barely. This might be because it is a really old game with weird graphics, and the panning in that game is also unusual. However, for the other games, the results showed a preference for the in-game panning. This is logical considering that in-game panning involves real-time 60FPS rendering, whereas my panning was at a rate of 2-3 seconds per frame and 5 degrees at a time.

However, regarding quality, variety, and aesthetics, the results were mixed. Some participants preferred the ground truth, while others chose AI-generated content. This part surprised me because I initially believed that no one would prefer the current state of AI-generated games over real games. However, seeing how AI can capture the essence of games, especially in the case of Minecraft, makes me wonder if, through fine-tuning and the use of multi-modals, we can already create feasible AI-generated image-to-image games.

When comparing my work with Adobe's Beyond the Seen 360° panoramic image generation and panning, I felt proud. Seeing my own result, I knew that the variety and quality were superior, and the user study confirmed it. Adobe's result, even though it is two years old, lacks in terms of quality and variety. The generated images don't look good, and the seam at the end of the panoramic view is glitched. The only advantage they provide is smooth panning, as the image is already generated and they just rotate around. However, even that seems to fail to capture the users' opinions. Surprisingly, my two-second-per-frame panning appears to be the user favorite!

In the absolute category rating part, it appears that the results are dependent on the image. While "hacettepe1" performed poorly, I assume it's due to its narrow panning, which caused hallucinations.

Users captured this as something undesirable. On the other hand, the "anime" and "ark" generations scored "good" overall. One of the most inspiring ones was "hacettepe3," where I achieved a complete 360° rotation. It transitioned from day to night, passing through flowers. However, since it didn't retain the time context, it scored low on context capturing and quality. Nevertheless, the seamless panning and the process of time passing, coupled with the bokeh effect from the flowers, seemed to be inspiring enough for users.

Participants were asked whether they believe the research can lead to creating games with AI-generated panning. The responses were promising:

- Strongly agree: 8 responses
- Agree: 10 responses
- Neutral: 6 responses
- Disagree: 2 responses
- Strongly disagree: 1 response

It appears that only a few people think this research might not reach its full potential. However, we all know that technology expands at unpredictable rates, and soon, with further developments, this research and work can indeed lead to fully 3D image-to-image generated games.

5 CONCLUSIONS

The analysis of the user study indicates that the AI-generated panning videos received a mixed reception across different attributes. While some videos were rated highly for their visual quality and context capturing, others lagged behind in realistic panning and inspiring AI. Pair comparisons showed a preference for AI-generated videos in some cases, but ground truth videos were still preferred in certain aspects, especially in terms of panning quality.

The overall participant sentiment suggests cautious optimism about the potential for AI-generated panning to enhance game development, though opinions varied significantly.

Based on the user study results, which indicated a stronger agreement towards inspiration, I believe that the created tool (shared via Google Colab) can already be utilized for content creation or for generating simple panning videos/GIFs for various platforms.

In conclusion I think this research was important step that hopefully will capture attention regarding creation of 3D realms from 360° panoramic image generation to another possible alternative of panning, warping with outpainting. As mentioned in the introduction, this method not only enables panning but also generates a flow of images that, when contextualized in a series, can provide a relaxing and immersive 3D experience. The shortcomings may be overcome with current technologies like GPT-4o or Gemini 1.5 multi-modals. Although I didn't have direct access to these technologies, I have seen demonstrations, and it seemed that whatever was missing from my research was already addressed in those functionalities.

In the short term, one can explore integrating warping into neural network structures and broaden the range of movement from panning to moving forwards and backwards. Additionally, step-by-step integration of user interactions into games can be pursued. This could begin with an AI that understands simple prompts like "open the door on the left" and evolve to more complex interactions such as "interact with the object in focus." The AI would comprehend concepts of interactions and be able to crop, move, modify, and copy objects in the image, creating a simulated 3D experience.

REFERENCES

[1] Brick2Face. (2023). Corridor Crawler Outpainting. Retrieved from https://github.com/brick2face/corridor-crawler-outpainting

[2] Midjourney. (2023). Documentation. Retrieved from https://docs.midjourney.com/docs/pan

[3] Tao Prompts. (2023). Create Panoramic Images With Midjourney's New Pan Feature! Retrieved from https://www.youtube.com/watch?v=P17BFXLHO0s

[4] Kevin Hohler. (2023). How to create 360 Panoramas using Artificial Intelligence (AI). Retrieved from https://blog.kuula.co/ai-panoramas

[5] Adobe. (2022). Project Beyond the Seen. Retrieved from https://labs.adobe.com/projects/beyond-the-seen/

[6] Chong Mou, Xintao Wang, Liangbin Xie, Yanze Wu, Jian Zhang, Zhongang Qi, Ying Shan, Xiaohu Qie. (2023). T2I-Adapter: Learning Adapters to Dig out More Controllable Ability for Text-to-Image Diffusion Models. Retrieved from https://arxiv.org/abs/2302.08453

[7] lkwq007. (2021). Stable Diffusion Infinity. Retrieved from https://github.com/lkwq007/stablediffusion-infinity

[8] lllyasviel. (2023). Fooocus. Retrieved from https://github.com/lllyasviel/Fooocus

[9] LykosAI. (2023). StabilityMatrix. Retrieved from https://github.com/LykosAI/StabilityMatrix

[10] OpenCV. Geometric Transformations of Images. Retrieved from https://docs.opencv.org/3.4/da/d6e/tutorial_py_geometric_transformations.html

[11] InvokeAI. (2023). InvokeAI. Retrieved from https://github.com/invoke-ai/InvokeAI

[12] Stability AI. (2024). Stable Diffusion 3. Retrieved from https://stability.ai/news/stable-diffusion-3

[13] Wikiquote. (2024). Carl Sagan. Retrieved from https://en.wikiquote.org/wiki/Carl_Sagan

[14] Kayla Akyüz. (2024). Generative Panning with Outpainting. Retrieved from https://colab.research.google.com/drive/1L3JyQjQB4qi6i2cZwJUOq8R17PvEJ7MJ?usp=sharing

[15] Kayla Akyüz. (2024). Final. Retrieved from https://mega.nz/folder/Yy4BASwJ#ii5Tk58DTu-r9j8b-QVodA

[16] Kayla Akyüz. (2024). User Study. Retrieved from https://docs.google.com/forms/d/e/1FAIpQLSchsZAZW8q1CHKjMYznc4TB9qQwC5HV7AVH87M58RqUsigfwg/viewform

[17] Kayla Akyüz. (2024). User Study Results. Retrieved from https://docs.google.com/spreadsheets/d/13IL-osnilGQ9F2BhWB1WKbPEhwUIJriDS7AKCDTCSKg/edit?usp=sharing

[18] Kayla Akyüz. (2024). User Study Results Charts. Retrieved from https://lookerstudio.google.com/reporting/b92cfc83-73f4-42a2-8ce0-fa1ddd45049e