XROG: An Interactive Object Generation System for Extended Reality

1st Nishan Shehadeh
AI in CPS CS 8395)
Vanderbilt University
Vanderbilt, USA
nishan.g.shehadeh@vanderbilt.edu

***Abstract*—This paper presents an interactive object generation system, XROG for extended reality, defined as the interaction between physical and virtual worlds through virtual reality or augmented reality. The system was designed and built on Microsoft's augmented reality headset the Hololens 2. It uses custom hand gesture recognition through real time hand tracking that captures sketches made by the user and uses them as input to a machine learning (ML) algorithm that classifies the sketch and informs the addition of a newly generated 3D object into the virtual scene. This interactive framework has a variety of applications in multiple industries and platforms from augmented reality (AR) environments where virtual objects are added to the physical world to virtual reality (VR) where the environment itself is completely virtual. The system is a novel attempt at integrating user input into object generation through custom hand gestures in real time. All project code can be found at: https://github.com/nshehadeh/AI_CPS**

***Index Terms*—extended reality, virtual reality, augmented reality, hand tracking, machine learning, object generation, Unity**

I. Introduction

The integration of machine learning networks with virtual environments offers tremendous potential for co-creativity between humans and artificial intelligence (AI). Recent developments in deep learning and XR headsets and environment development provide a host of new opportunities to explore different forms of interaction and co-creativity in virtual environments. In its simplest form, user input defining the introduction of virtual objects adds a new level of creative ability on top of normal object manipulation or game-like mechanics of virtual objects.

In this project, Interactive Object Generation System for Extended Reality (XROG), a system was developed for user generation of virtual objects that integrates custom object generation into a virtual scene in real time and acts as a framework for future work in generating virtual content as object generation models become more accessible.

A. Motivation

The development of an interactive object generation system, as described in this paper, has the potential to transform a wide range of industries, including gaming, architecture, interior design, education, medicine, and other types of training. In gaming, this system could be utilized to create new and immersive gameplay experiences. For example, an application giving users similar abilities to the Green Lantern character where they can generate objects to use as part of combat gameplay. Players could create unique and customizable objects, such as shields or weapons, that they could use in battle. Additionally, adding interactive object generation to gaming worlds similar to Minecraft could allow users to creatively build and design structures as part of the gameplay. Players could use the system to generate 3D objects in real-time and incorporate them into their in-game creations, allowing for endless possibilities and creativity.

Industrial designers could also benefit from this system by using it to create and visualize 3D models of products and prototypes. The models could be manipulated in real-time, allowing for adjustments and improvements to be made to the design as necessary. In the realm of marketing and advertising, this system could be used to create interactive product demonstrations and advertisements. Customers could see how products work in 3D and make informed purchase decisions.

The system could also have applications in education and training. Interactive 3D models could be created for teaching and learning purposes, such as medical students studying anatomy or surgery in 3D or mechanics visualizing complex engine parts.

Overall, the proposed system has the potential to impact many industries by offering a novel approach to 3D object generation and visualization. By blurring the line between the physical and virtual worlds, this system could transform the way we interact with our surroundings and change the way we work, learn, and play.

B. Challenges

There are many challenges with creating a system like XROG. Because the system integrates many different types of systems such as hand tracking, ML classification, and virtual object generation, each module must be incorporated in a manner that allows the system as a whole to operate seamlessly

in real time. Individually, these parts are all areas of research within the XR and broader community. Specific challenges will be discussed within the respective implementation sections for each module within XROG.

C. Novelty

XROG's main novel contribution is the end-to-end nature of the system that allows for all stages of training and inference that can be combined into a complete system. Additionally, individual aspects of XROG offer novelty in how it uses Hololens hand tracking through custom handling of the data for ML and gesture recognition.

D. Contributions

- Implemented a system for data collection of 3D sketches on a Hololens that can be applied to a variety of other XR devices. The sketches are processed in a way that is usable for ML training and inference.
- Trained a ML model to classify input sketches in real time (sparse 3D point clouds) using an SVM.
- Built a Unity scene that provides a basic game with three potential 3D objects that the user can generate through sketching.
- Integrated all parts of XROG into a system that allows a user to generate virtual objects in real time using a cloud-hosted classification model and responsive Unity environment on the Hololens 2.

II. Related Work

Since this project consisted of designing a system/experience as opposed to implementing a new type of network, there is a variety of related work including papers that describe similar types of overall systems, gesture and hand-tracking papers, and 3D model generation papers.

A. Gesture and Sketch Recognition

The main task accomplished in this AR system was a form of gesture or sketch recognition network that can take what a user is drawing and add objects to the virtual scene based on the input. There are a few different paths of related work regarding user input in the environment. Sketch based image retrieval (SBIR) deep learning methods involve networks that can take 3D or 2D sketches and classify objects based on them. Alternatively, computer vision models recognize gestures as patterns of hand movements and then classify them as objects. There are many networks considered state of the art for both approaches. SketchANet builds on Alexnet to perform sketch recognition [9]. There are also a variety of other CNN-based approaches including Siamese CNNs like the one proposed by Koch et al. that use compatibility between images and sketches using edge-maps [4]. Lastly, triplet networks have shown promise in the SBIR field. Bui T et al. were the first to propose a triplet architecture that use positive and negative examples to relate sketches and objects [2]..

B. 3D Object Generation

OpenAI currently has the cloest thing to real time 3D object generation through its Point-E network [6]. Point-E is designed to quickly create point clouds from text input through diffusion models. It is publicly available on HuggingFace and GitHub. Importantly, this system is designed to be light-weight and only take 1-2 minutes to create models on a single GPU compared to hours with other state of the art 3D generation systems. Other 3D generation systems that have had success include NVIDIA's instant NeRFs that uses neural radiance fields to convert imagery into 3D models [5]; GPT-3D which uses transformer-based architecture to generate 3D objects from textual descriptions [1]; and StyleGAN2 which uses GANs to generate realistic 3D models [3]. Although these models represent the state of the art in the field, they require a large amount of compute and time to run. Although my implementation of XROG does not use a generative model in this iteration, the framework is there to include these types of models in future iterations when they are more lightweight and accessible.

C. Generative AI in XR

This section of related work will focus on other descriptions and attempts at designing systems that promote co-creativity in virtual environments. In "Designing Co-Creative AI for Virtual Environments", the authors describe a framework for co-creative AI systems in virtual environments [8]. They made a system called Calliope that enables users to explore and manipulate generative design solutions in real time through a process where the user "sculpts" meshes from scratch using drawing motions or generates objects from a menu. This shares many similarities with my proposed system; however, it focuses more on mesh manipulation instead of object generation. Similarly, Ratican et al. propose a pipeline that describes overall possibilities regarding generative AI models in extended reality content for the metaverse [7]. They cite many important motivations for this type of pipeline including content creation for metaverse applications and democratizing 3D design and modelling to allow people with a variety of experience to develop virtual content. Although neither of these papers design an end-to-end systems, they propose a wide range of motivations and applications for this type of system.

III. System Design

XROG consists of two main parts: training and inference. In training, data is collected and the classification model is trained. In inference, users can use the cloud-hosted classification model to generate objects in the scene. In the proof of concept implementation, the Unity scene was a simple scene with a training "dummy soldier" virtual object. The user was able to draw and generate three objects: a sword, a shield, and a star.

Fig. 1. Overview of the training module

A. Training

In the training portion of the system, data is collected from the Hololens through a unity environment and saved out in CSV format before being used to train the classification model. The Unity environment responsible for allowing the user to create a dataset of gestures involves a script that records the location of the user's index finger while they are pinching. More details can be found in the implementation section. Training was run twice for XROG: the first time with around 35 samples collected per object class and the second time with around 100 samples collected per object class. After collection, the samples are saved and loaded into a ipynb file to be preprocessed and train an SVM using sklearn.

B. Inference

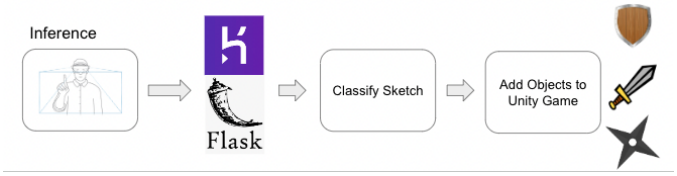

Fig. 2. Overview of the inference module

In the inference portion of the system, the user actually sends real time data to the cloud app hosting the trained classification model in order to interact with their environment in a creative way. The user sketches the object in 3D and the data is uploaded to a web app hosted in the cloud where classification takes place. The web app then sends the prediction results from the model back to the Unity scene, and the corresponding object is loaded in for the user to interact with. More details on how this was implemented will also come in the next section.

IV. Implementation Details

A. Training

1) Data: The data collection, as outlined before, was done through Unity and Microsoft's Mixed Reality Toolkit (MRTK). MRTK offers hand tracking support which can automatically track the location of the palm and key digits like the index finger and thumb as seen in Figure 3. It also can recognize when a user is pinching (connecting the index finger and thumb together). In order to make a more user-friendly experience for data collection, XROG uses the pinching motion as the beginning of data collection for a sample. Upon pinching the fingers, a red sphere appears at

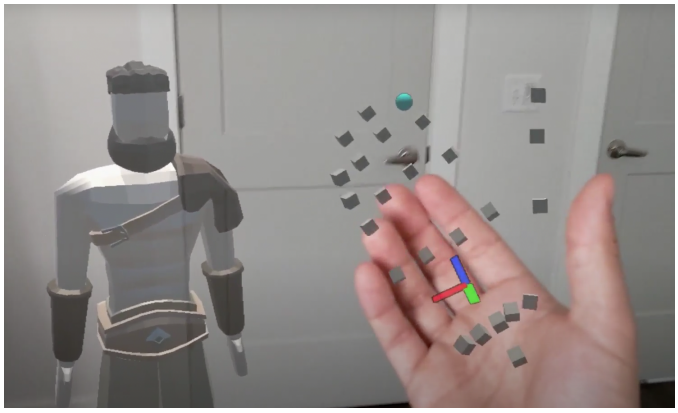

Fig. 3. Standard MRTK hand tracking

Fig. 4. MRTK hand tracking with red sphere indicating data collection

the connection of thumb and index finger to help the user identify that they are collecting a sample as seen in Figure 4. While pinching, a vector of coordinates is saved for the index finger into a sample array. When the user is done drawing and un-pinches, the array is saved out to a CSV. Following this format, the user can make many samples in a short amount of time to create a properly sized dataset for classification. Once saved out, each sample has a sample number and then of shape {frame number, x position, y position, z position}. These samples can either be used with their frame number as temporal data or as point clouds where the frame number is not important. For this iteration, the temporal data was not important for classification, so the data was treated as a sparse point cloud, exported, and preprocessed using python. Figure 5 shows an example of a plotted point cloud collected. For the three classes used (shield, swords, and stars), a gesture was arbitrarily assigned to each class. The shield was gestured for by drawing a circular shape, the sword was gestured for by drawing a cross shape, and the star was gestured for by drawing a 5-pointed star. These can all be viewed in the video demonstration on this project's github.

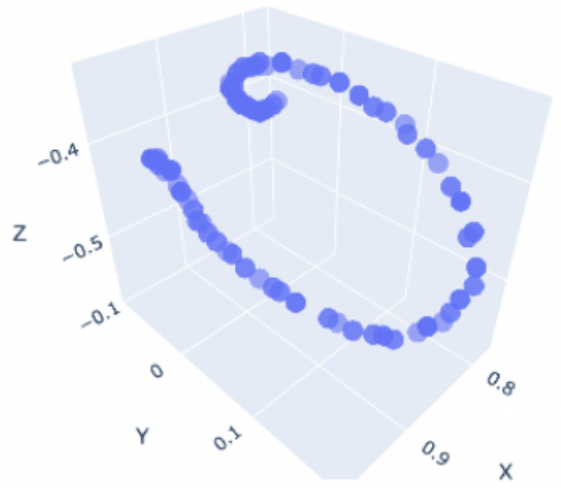

Fig. 5. Example sample displayed using plotly

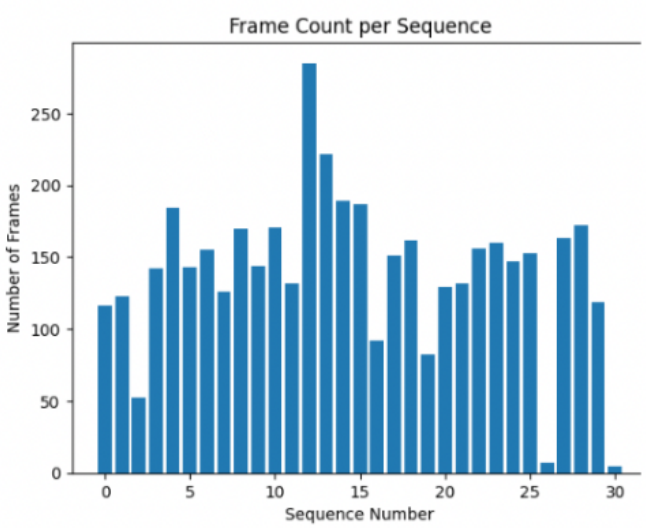

Fig. 6. Number of frames for 30 samples of

2) Preprocessing: The preprocessing necessary to make the data usable for classification done by a ML model can be split into three parts 1) resampling all point clouds such that they have the same number of points, 2) centering and normalizing the point clouds, 3) performing data augmentations.

In order for the point clouds to be used as input for an ML model, they all must have the same dimensions. Given the data, which can be plotted in 3D as in Figure 5, is raw output from hand tracking as described earlier, each 3D sketch is collected for a different number of frames (based on how long the user took to draw the sample). While collecting data for XROG, a range of around 1s to 5s was used to collect the 3D points. Given a target framerate of 60s, this leads to a large variety in the number of points for each sample point cloud, as shown in the graph in Figure 6. A reference number of goal sample points was found by first removing all ”short sequences” such as sequence 26 in Figure 6 that were a result of a user quickly or accidentally pinching leading to a failure to collect a proper point cloud. This cutoff was set to 30 frames, or, theoretically, half a second of pinching. The average frames per sample was then computed, and each sample was upsampled or downsampled accordingly. For upsampling, nearest neighbor upsampling was used where the density of the point cloud is increased to the target size by replicating points in the original cloud. For downsampling, random sampling downsampling was used where points are randomly selected from the original cloud, resulting in a smaller point cloud with a reduced number of points but preserving the overall structure and characteristics of the original cloud. Once all point clouds

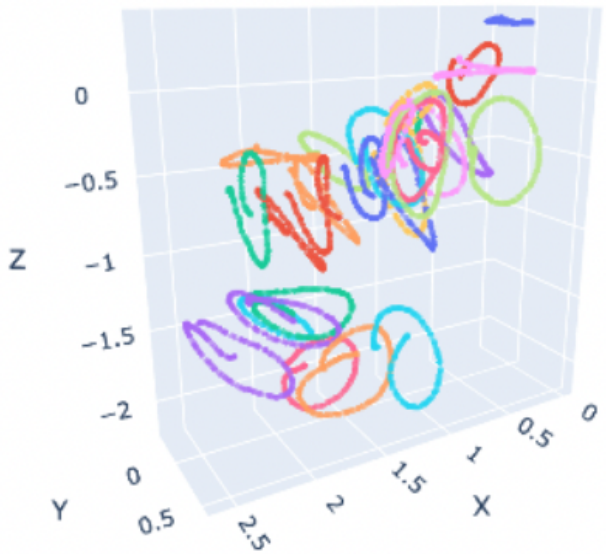

Fig. 7. All samples for the shield class in 3D space after being resampled

were of the same density, an example of which is shown in Figure 7, or same number of frames, they were all combined to one dataset with their corresponding class labels. Each class was equally represented in the dataset by taking the first x samples of each class where x is the number of samples in the smallest class. The samples were then all aligned around the origin point and rescaled as shown in Figure 8. The last step in the preprocessing pipeline was to perform data augmentations. Translation, rotation, and noise augmentations were applied to create an additional 10 augmented samples for every individual sample. These data augmentations are useful for sparse point cloud classification because they increase the variety of the training data, which can improve the performance of machine learning models. The translation augmentation adds random translations to the point cloud, which can simulate changes in

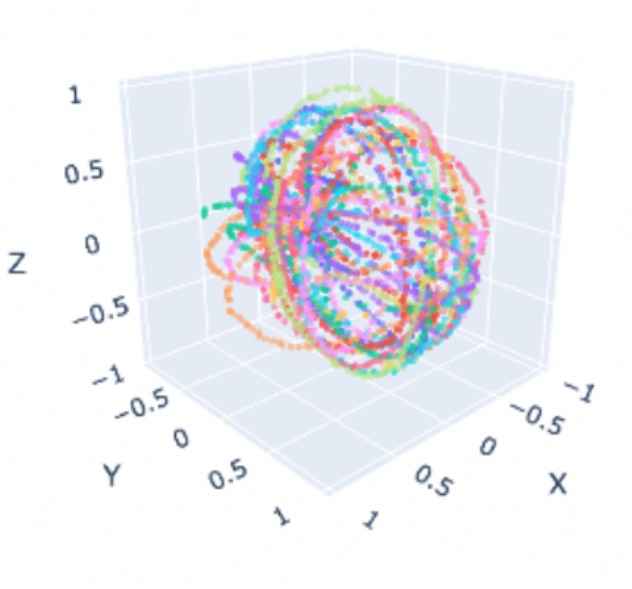

Fig. 8. All samples for the shield class in 3D space after being aligned and rescaled

viewpoint or object position. This helps the model learn to be more robust to changes in the spatial arrangement of the points. The rotation augmentation applies random rotations around the x, y, and z axes to the point cloud, which can simulate changes in object orientation. This helps the model learn to be more robust to changes in object pose. Finally, the noise augmentation adds random Gaussian noise to the point cloud, which can simulate sensor noise or measurement errors. This helps the model learn to be more robust to noise in the input data. By using these data augmentations, the model can learn to generalize better to unseen data and improve its ability to classify sparse point clouds.

B. Inference

1) Data: At inference, the data is captured in the same method described before for training. This time, each sample is immediately uploaded to the web hosted python app for classification instead of being added to a CSV file and saved out. Similar preprocessing is done to the sample so it is in the same format as the point clouds used to train the model. That is, the point cloud is resampled, aligned, and rescaled using the same techniques as above. The data is then fed through the machine learning model and its classification results are returned to the Unity scene as described in the next two sections.

2) Cloud ML: Because Unity is built on C# and does not have good integration for python models, XROG uses cloud computing to host the machine learning model and perform classification remotely. This hosting was done using FLask and Heroku. Flask is a lightweight, open-source Python web framework that allows developers to build web applications quickly and easily. Heroku is a cloud-based platform as a service (PaaS) that enables developers to deploy, manage, and scale their applications easily and quickly. Heroku supports a wide range of programming languages including Python, Ruby, Java, Node.js, and more. It provides developers with a streamlined deployment process, automatic scaling, and a range of tools and services to help them build and manage their applications. To use Heroku, XROG simply uploads its inference code to the platform, and Heroku takes provides the rest of the necessary services including provisioning servers, scaling, load balancing, and more. The inference python file loads in the model checkpoints outputted after training so it can access them during inference. The file receives the sample through a HTTP post in json format before processing it, classifying it, and returning an integer corresponding to the class of object that the sample belongs to according to the model.

Fig. 9. Examples of generated objects in the Unity scene

Fig. 10. Action example of a user throwing a generated ninja star (star class)

3) Unity Scene: The Unity scene itself is a fairly simple game environment as described previously where there is a dummy solider and the user is able to generate swords, stars, and shields depending on their sketch, designated by a pinching motion. The objects that were generated in are low poly prefabs taken from the Unity Asset store. They are fully interactable as you can see in Figure 9 and Figure 10. There is support for MRTK's near and far interaction for the Hololens.

V. Evaluation

A. Sketch/Gesture Classification

Classification accuracy was not a main focus of this project. Because there are few other projects building custom gesture classifiers at all for the Hololens, there are no baselines to compare my results to. My goal was to build a system that works in general, so the results are main qualitative as described in the next subsection on the scene's evaluation. For the second

Classification report:

	precision	recall	f1-score
0.0	0.86	0.86	0.86
1.0	1.00	1.00	1.00
2.0	0.89	0.89	0.89
accuracy			0.89
macro avg	0.92	0.92	0.92
weighted avg	0.89	0.89	0.89

Accuracy: 88.89%

Fig. 11. Classification report for a sklearn SVM trained on the larger dataset

dataset collected and used in XROG's classification, there were on average 100 samples collected for each class. These were augmented into 1010 samples for each class based on the augmentation techniques previously described. The SVM used a linear kernel on the flattened data (getting rid of frame number) with a regularization parameter of C = 1.0. It was able to achieve an accuracy of just under 89% with a full classification report as listed in Figure 11.

B. Unity Scene

Qualitatively, the Unity scene was very successful in generating objects in real time. XROG was not tested for any formal qualitative feedback on blind users, but it was able to generate objects in real time in an intuitive way. There were no major bugs or flaws with the program. Although there were occasionally objects accidentally generated by pinching being picked up by the hand tracking that was not meant to be a real pinch, through testing it seemingly got object classification correct every time. In the official github repository linked in the abstract, there is a demonstration video for the environment that shows XROG's capabilities in real time.

VI. Discussion

A. Limitations

The main limitations for XROG is its simplistic ML model and lack of built out Unity scene. Additionally, the cloud computing used in this system is overall quite low tech, and given enough back and forth between the Unity scene and a more intensive model, it may succumb and slow down the scene's ability to react to input tremendously. These limitations will be further addressed in the future work.

B. Future Work

1) Improvements on Current Implementation: The main avenue for improvements on XROG's current implementation is through building out the actual Unity game and expanding into another VR environment to show its capabilities in other forms of XR. Currently, there is no real game logic or game functionality and the environment is simply showing off the ability to add custom objects to the scene. Also, the number of objects the user can generate is extremely small (3), so, in order to further test XROG's capabilities, it is necessary to expand the possibly objects that can be generated. This expansion will also test whether classification models can keep up with gestures that will inevitably get more similar to each other and harder to differentiate for the ML model. As discussed in the related works section, there are other classification networks that can also be experimented with including vision based and point cloud based deep learning techniques.

2) Long-term Possibilities: For long-term possibilities of XROG, there are many avenues that a system like this can become prevalent as generative AI and XR experiences become exponentially more popular. The ability to generate virtual content of any kind, from objects to entire environments open up a new type of possibly computationally creative environments. A framework like XROG is proof of concept that it is possible to build a system that allows for this type of interaction between users' input, machine learning results, and virtual content.

VII. Conclusion

In conclusion, the interactive object generation system for extended reality (XROG) presented in this paper offers a novel approach to 3D object generation and visualization. By using custom hand gesture recognition and machine learning algorithms in real time, the system allows users to generate and manipulate virtual objects in augmented or virtual reality environments. This framework has a wide range of potential applications in various industries such as gaming, architecture, interior design, education, medicine, and training. XROG's main novelty lies in its end-to-end system that seamlessly integrates hand tracking, machine learning, and virtual object generation. The system's contributions include a data collection system, a real-time sketch classification model, and a responsive Unity environment on the Hololens 2.

REFERENCES

[1] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[2] Tu Bui, Leonardo Ribeiro, Moacir Ponti, and John Collomosse. Compact descriptors for sketch-based image retrieval using a triplet loss convolutional neural network. *Computer Vision and Image Understanding*, 06 2017.

[3] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018.

[4] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.

[5] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Trans. Graph.*, 41(4):102:1–102:15, July 2022.

[6] Alex Nichol, Heewoo Jun, Prafulla Dhariwal, Pamela Mishkin, and Mark Chen. Point-e: A system for generating 3d point clouds from complex prompts, 2022.

[7] J. Ratican, J. Hutson, and A. Wright. A proposed meta-reality immersive development pipeline: Generative ai models and extended reality (xr) content for the metaverse. *Journal of Intelligent Learning Systems and Applications*, 15:24–35, 2023.

[8] Josh Urban Davis, Fraser Anderson, Merten Stroetzel, Tovi Grossman, and George Fitzmaurice. Designing co-creative ai for virtual environments. In *Creativity and Cognition*, CC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[9] Yongxin Yang and Timothy M. Hospedales. Deep neural networks for sketch recognition. *CoRR*, abs/1501.07873, 2015.