3D Unity Scanner

Mark J. McCulloh, Timothy C. Flowers, Christopher E. Williams, and Brandon J. Aulet

Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — In order to reduce the time, effort, and technical knowledge that students require to create video game levels, we have created a plugin for the Unity game engine that can streamline the design process. Our plugin allows students to use an RGB-D camera to scan blocks into Unity and immediately begin working on them.**

***Index Terms* — Computer vision, digital images, image analysis, Image color analysis, object detection.**

I. Introduction

A common practice in game design is to physically model a level before digitally recreating it by manually placing objects in their game engine of choice. Depending on the complexity of the level, this can be a multi-hour endeavor. We aim to reduce the time it takes to create a playable level based on a physical prototype.

We created a Unity plugin that allows users to scan blocks directly into the game scene. The blocks are manifested as Unity Game Objects, thus allowing users to immediately begin working with them to create a game. A user can have a playable level with a single scan with a depth camera.

Currently, there are some techniques that can scan and model objects in real time. The issue with that however, is that most of those scanners either don't segment individual objects or they require specialized hardware that isn't readily accessible to students. Identifying separate objects is very important for the purposes of game design since each block might have its own texture or logic. If only one model is created for a group of blocks, it would require much more time and effort to manipulate in the game engine.

The amount of similar software dwindles even further when you add the fact that this needs to be a Unity plugin. On the Unity Asset Store, there are no examples of plugins that can be considered similar to the kind of plugin that we have created.

II. Problem Statement

Our group was asked by the Games Research Group (GRG) of University of Central Florida to create a plugin for the Unity game engine that would allow users, students in this case, to scan blocks into the game scene.

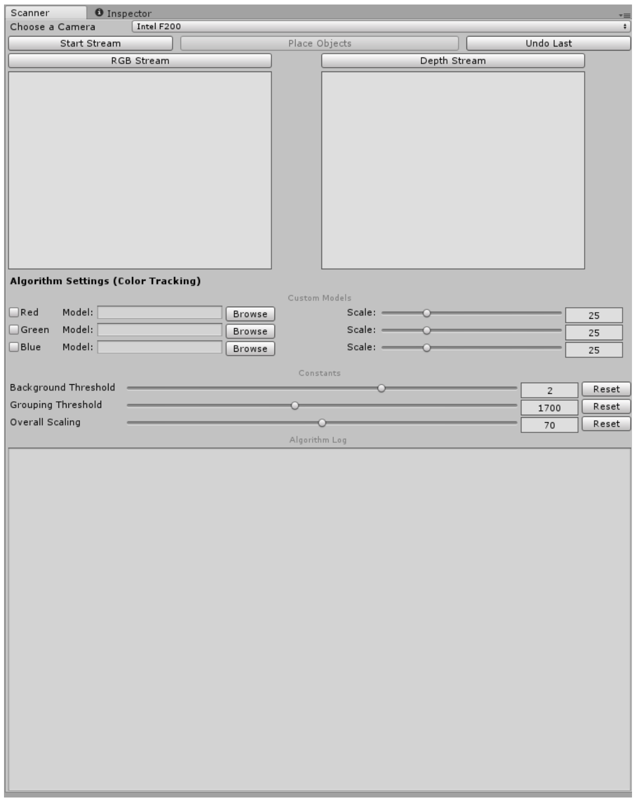
The process of creating a video game level can be very arduous. The designer needs to make sure that the level flows and feels right spatially. It is much easier to do so if one builds out a level physically, using paper models or blocks, and once it is finished then it can be rebuilt digitally so that it can be textured and have game logic applied.

This in turn not only affects students, but also other aspiring game designers or simulation creators that would like to expedite their production of prototypical video game levels in Unity.

III. Experimental Study

Originally, we researched cutting-edge methods that implemented powerful computer vision systems and data structures. These included many conference papers from CVPR 2016 including *Uncertainty-Driven 6D Pose Estimation of Objects and Scenes from a single RGB Image* by Brachmann et al. which utilized a combination of a proprietary auto-context random forest structure with Random Sampling Consensus (RANSAC) for pose estimation [2]. Many of these advanced detection algorithms required access to an annotated dataset of 3D scenes and models for training, which led us to research RGB-D datasets like *A Large-Scale Hierarchical Multi-View RGBD Object Dataset* by Xiaofeng et al [3]. We also looked into the simpler, but still fairly accurate, POSIT algorithm [1].

These methods would have allowed us to more accurately determine the position and rotation of the blocks. We attempted to incorporate these methods into the plugin using libraries and frameworks like the Accord.NET framework [4]. Unfortunately, the limitations places upon us by Unity made implementing these algorithms infeasible for our timeframe.



Choose a Camera: Intel F200
Red Model: Scale: 25
Green Model: Scale: 25
Blue Model: Scale: 25
Background Threshold 2
Grouping Threshold 1700
Overall Scaling 70

IV. Plugin Structure

Our plugin is comprised of three main modules: the Camera Interface module, Unity Module, and Image Analysis Module are described below in sections A, B, and C respectively.

A. Camera Interface Module

The camera module's sole task is to obtain data from the depth camera and make it available in a common format that Unity can easily utilize. Accessing a camera's data is abstracted by the interface ICamera. By accessing the camera data through the ICamera interface, the plugin can request data from a user selectable camera without having to be aware of the exact implementation of the process for acquiring a camera from the image.

The ICamera interface has five method members: StartCapture, GetImage, SetImage, Get3DPointFromPixel, and StopCapture. StartCapture is called before image capture takes place and is responsible for performing any initialization necessary for the camera to begin capturing images. GetImage is called once for each image that the plugin wishes to process and is responsible for: acquiring image data from the camera, converting the data to Unity's Texture2D type, and releasing any resources that were required to acquire the image. The GetImage method returns a ColorDepthImage object which contains a Texture2D object representing the color image and a Texture2D object representing the depth image. SetImage is called by the image analysis algorithm being used and takes a ColorDepthImage as a parameter and stores it to perform the projection operations in the Get3DPointFromPixel method. Get3DPointFromPixel is called by the image analysis algorithm to perform the conversion from the uvz coordinate system to the xyz coordinate system. StopCapture is called once the plugin is finished acquiring images from the camera and is responsible for releasing any resources required to communicate with the camera.

Using the comprehensive ICamera interface, the Unity Module can not only interface the camera without knowledge of its implementation, but the plugin can also be easily expanded to support other cameras by creating additional implementations of the ICamera interface.

The implementation of ICamera that we provide is for the Intel RealSense F200. It utilizes the Intel RealSense SDK to provide the implementation details of the ICamera method members.

B. Unity Module

The Unity Module is responsible for creating the user interface of the project, as well as linking the other two modules together and populating the game scene with the new GameObjects. The module is encompassed in Scanner.cs, which is a C# script that derives from Unity's EditorWindow class allowing it to show up as a custom window in Unity titled 3D Scanner.

The custom window has a dropdown menu allowing users to choose from the implemented cameras available. Currently, the two options are the Intel F200 and Dummy Camera (used for testing), this was implemented with the thought of other cameras being added to the plugin at a future time.

Once a camera has been chosen, the user can start the capture and use the live feeds to determine the optimal position for the image capture. Users can toggle between a RGB stream or "cool" stream and a depth stream or processed (contrast) stream through the two windows. Once a suitable angle has been found the users can click on the "capture" button which will send the captured image to the image analysis module.

Once the analysis is complete, a list of GameObjects is returned to the Unity module. First, a primitive plane is populated to represent the "floor" of our level. Since it too is a GameObject, the user can delete it in seconds if they decide it's not needed. After the plane is created, the list of GameObjects is iterated through and each object in the list is assigned a color based on the kind of block it is before being populated in the game scene. Right after the block is populated the object data is printed on the log and the user can then decide to undo and try again or keep the scanned blocks and begin working with them.

The Unity module also offers a few more advanced options for the user. The standard deviation and clump threshold settings help the user configure the algorithm if needed, under suitable lighting default values are recommended. The three color tracking fields allow the user to choose a model to replace different colored blocks with. For example, if a user wanted to place previously modeled Non-Playable Characters in the scene, they can select the model in one of the file pickers and all corresponding blocks of that color would be swapped out for the chosen model.

C. Image Analysis Module

The Image Analysis Module is responsible for analyzing the data acquired by the Camera Module and producing the GameObjects to populate the Unity Scene with. The Image Analysis Module is abstracted by the IAlgorithm interface. By abstracting the analysis through the IAlgorithm interface, future algorithms can be implemented to perform different types of analysis. This coupled with the ICamera interface of the Camera Module, allows for any camera to provide the data for any image analysis that is implemented for the plugin.

The IAlgorithm interface consists of five method members: DrawSettings, PreviewImage, ProcessImage, GetShapes, and ClearShapes. DrawSettings is called by the Unity Module's OnGui method and is not only responsible for drawing the user interface which is specific to a specific implementation of the IAlgorithm interface but also saving the values that the user interface is used to capture from the user. PreviewImage is a method which takes a ColorDepthImage as a parameter and returns a Texture2D image. The Texture2D image which is returned, represents intermediate processing performed by the image analysis algorithm in real time. This is presentable to the user so that an ideal positioning and framing of the image can be chosen to be used for the full image processing. ProcessImage is a method which takes a ColorDepthImage as a parameter. The method is responsible for performing the full processing on the image and saving the results. The ProcessImage does not return the final results to allow for algorithms which compile final results from the processing of multiple images. GetShapes is a method which returns an enumeration of GameObjects to be populated into the Unity scene. ClearShapes is a method which resets the state of the algorithm back to its initial state with no images processed and no results yet calculated.

Our specific implementation of IAlgorithm was the ColorTrackingAlgorithm class. The underlying algorithm works by isolating the blocks within the image from the background/table and mapping each of the distinct blocks to a user chosen GameObject based on the color of the block. Our algorithm assumes that the background/table is relatively monochromatic and that the blocks on the table are red, green, or blue.

When ProcessImage is called, first the algorithm determines the color of the background by calculating the average color of the entire image. Then the standard deviation of the average color is calculated. The difference of two colors is calculated as the distance between the average color and a given pixel's color as if the two colors were three dimensional vectors composed of red, green, and blue components. Using the average color of the image and the standard deviation of the average color, a refined average color is calculated by only considering pixels which fall within a user-specifiable number of standard deviations from the average color. After the refined average color of the image has been calculated, a new version of the image (called the contrast image) is created using only black and white pixels. All the pixels of the original color image which fall within a threshold distance of the refined average color are made white in the contrast image. All pixels from the original color image which fall outside of the threshold distance from the refined average color are made black in the contrast image. Once the contrast image has been created, an iterative flood fill algorithm is used to combine adjacent groups of black pixels into organizations called clumps. Once a clump has been created the average color of the pixels which compose the clump is calculated using the original color image. The clump object, which includes the pixels which compose it as well as the average color is placed into a queue. The table normal is then calculated to determine what angle the camera is viewing the block scene on the table from. To calculate the table normal by sampling 11 random pixels that have been marked as background pixels. One pixel is chosen to be tail of all vectors and the remaining 10 are the heads of the generated vectors. The process produces 10 vectors which are parallel to the table. We then calculate all possible pairing which do not consist of the same two vectors. We calculate the cross product of all such pairings and take their average by summing the cross products and normalizing them.

When GetShapes is called on the ColorTrackingAlgorithm object, the queue of clumps begins to be dequeued. Each pixel is mapped from uvz coordinates to xyz coordinates to create a three-dimensional point in space. The average position of all the three-dimensional points created from a clump is then calculated. This point is then rotated about the x-axis by the angle between the table normal vector and the vector that represents the perspective of the camera. The final three-dimensional point is then used as the position of the GameObject which will represent the clump and the block that it corresponds to from the image. The dominant color channel of the average color of the clump is then used to determine whether the block was red, green, or blue. A GameObject of the model type which the block color maps to is then created at the calculated average position. A sequence of these blocks is then returned to the caller of the GetShapes method.

V. Conclusion

Although it cannot be said that we have successfully implemented a plugin capable of scanning entire levels into Unity, we have made a large amount of progress on research towards that goal. On a single vertical layer, we can scan in blocks with accuracy and can swap them out with custom models. The camera module and image analysis module were written with the thought of upgrading them in mind, making it easy for future developers to add functionality to the plugin.

Acknowledgement

The authors wish to acknowledge the assistance and support of Dr. Mark Heinrich for guiding and advising us throughout the duration of out project.

We would also like to acknowledge our sponsors Dr. Peter Smith and Matt Dombrowski for providing us with the idea and the tools necessary to create our plugin.

References

[1] D. F. DeMenthon, and L. S. Davis, “Model based object pose in 25 lines of code,” *Proceedings of Second European Conference on Computer Vision*, pp. 112-117, 1994.

[2] Alexander Krull, Michael Ying, Yang Stefan, Gumhold Carsten Rother, Eric Brachmann, Frank Michel. “Uncertainty-Driven 6D Pose Estimation of Objects and Scenesfrom a Single RGB Image,”2016, http://wwwpub.zih.tu-dresden.de/~cvweb/publications/papers/2016/rgbpose.pdf.

[3] Xiafeng Ren Kevin Lai Liefeng Bo and Dieter Fox. “A Large-Scale Hierarchical Multi-View RGB-D Object Dataset”. In *ACCV(2012).*

[4] Accord.NET. Accord.NET Framework Documentation. 2016. http : / / accord-framework.net/docs/html/R_Project_Accord_NET.htm.