# Volumetric Painting in Virtual Reality using voxels

ANUBHAV CHAUDHARY; 2016013 and YASHIT MAHESHWARY; 2016123

**ACM Reference Format:**
Anubhav Chaudhary; 2016013 and Yashit Maheshwary; 2016123. 2019. Volumetric Painting in Virtual Reality using voxels. 1, 1 (April 2019), 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Github Link

## 1 INTRODUCTION

With Virtual Reality digital painting is being expanded from drawing on 2D surfaces to drawing in a 3D world. This paper shows a novel technique to allow creation of graphic models in 3d with voxels rendered in real time. [1]

## 2 LITERATURE SURVEY

There have been several solutions presented by various authors in this field. The current paper presents a novel technique of generating models on a very large canvas (about $40km^3$) with very high details. There are also techniques which use octree as a way of representing spatial 3d voxel grids so as to improve performance. An octree representation is good since the tree can be refined at any depth as per requirement. To reduce this traversal time, a more shallow tree has been used. [3]

Another improvement which we found during literature review involves marking part of scenes as dynamic and static. Static and dynamic parts are also stored in different parts of memory in the GPU to further improve performance. [2]

## 3 ALGORITHM AND GAME DESIGN ASPECTS OF THE PROJECT

The algorithm starts with creating a **chunk**. A chunk is a collection of cubes/voxels which are rendered in a 3d grid in order to generate a model/terrain. Each cube has 6 faces and can have 6 possible neighbours. In order to improve the frame, rate only those faces of a cube are rendered which are exposed to the environment and not covered by a neighbouring cube/chunk. The **world** is made using a collection of chunks. Each world has a 3d array of chunks which are used to generate a model/terrain. In the current version of the project we are using perlin noise to procedurally generate a terrain. Now once a model/terrain has been generated we have to give the user the ability to add/delete voxels so as to edit/remodel the object as per his/her requirement. The way this is done is by casting a ray from the camera to where the mouse points in the game view. There are two possible cases now

(1) The ray intersects a virtual object in the 3d scene. Then the intersection point of this ray is calculated and a new voxel is created at that point.

Authors' address: Anubhav Chaudhary; 2016013; Yashit Maheshwary; 2016123.


XXXX-XXXX/2019/4-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

(2) The ray does not hit any virtual object but travels a certain distance (configurable by the user) the final end point of the ray is used as the center point of the new voxel to be generated.

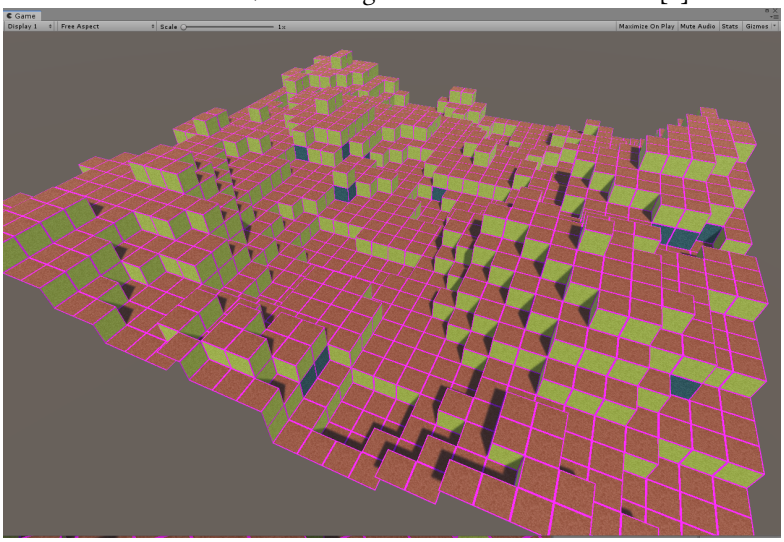
Now since we can't update the whole world on every addition/deletion as this would be very inefficient. Whenever a block is added/deleted we go over its neighbouring blocks marking them "to be updated" so that we don't update the whole world. At the end of each frame every block which has been marked is updated and by the start of next frame we have a new world which has been updated as per the user inputs.

## 4 A SHORT DEMO VIDEO LINK

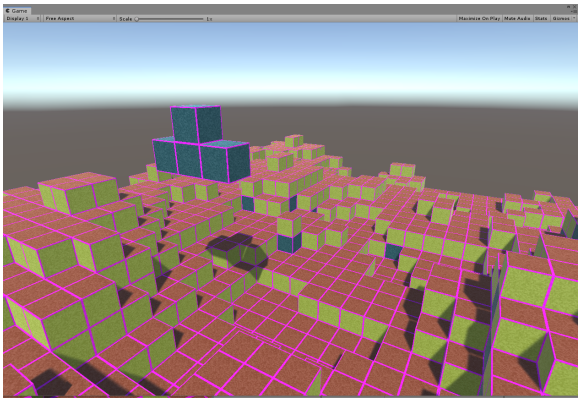
Uploaded along with the PDF Report on the classroom submission.

## 5 RESULTS

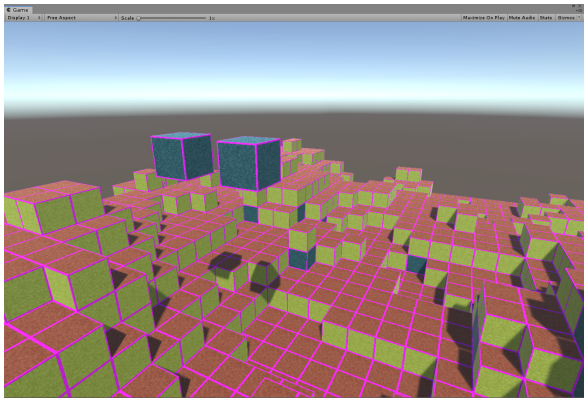
The source image of texture used in blocks/voxels is given in references section [4].



A sample terrain generated using voxels



Adding new voxels



Removing existing voxels

## 6 MILESTONES FOR FINAL DELIVERABLE

(1) Integrate VR controllers to facilitate easier input.
(2) Allow selection of voxel colors from a predefined set of colors.
(3) Allow placing object models into the scene (that would function like voxels).

## REFERENCES

[1] Yeojin Kim, Byungmoon Kim, Young J. Kim [*Dynamic Deep Octree for High-resolution Volumetric Painting in Virtual Reality*].
[2] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN [*Interactive indirect illumination using voxel cone tracing*].
[3] LEFOHN A. E., SENGUPTA S., KNISS J., STRZODKA R., OWENS J. D. [*Glift: Generic, efficient, random-access gpu data structures.*]
[4] Google Images