Implementation Plan

The project runs on two main components:

-Graphics: how the scene is rendered

-Algorithm: the animation implementation

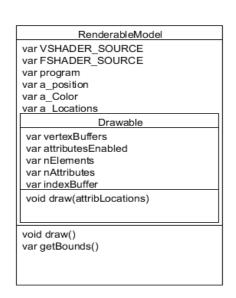
Plan A

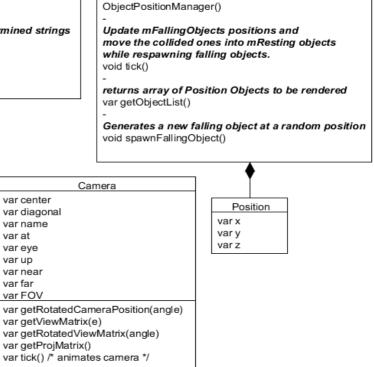
Our original plan was to split the project up into the following components and work on them individually:

ObjectSelecter, ObjectPositionManager, Camera, RenderableModel and a global thread to tie them together

Global Thread: var teapotObject var cubeObject var skullObject var ObjectSelecter var ObjectPositionManager var Camera gets each object position from ObjectPositionManager and renders a RenderableModel at that position tick()

	ObjectSelecter
	This Class is used to control which type of object
	is being rendered (cube, skull, teapot etc)
	object which contains the meshes/verteces etc mCurrentObject
_	
Con	nstructor
void	ObjectSelecter()
-	•
	tches mCurrentObject based on predetermined strings switchTo(string)
-	
retu	ırns mCurrentObject
varg	getCurrentObject()





3D array of objects at rest var mRestingObjects 1D array of falling objects var mFallingObjects

Constructor

ObjectPositionManager

Plan B

After we began the implementation, it was clear that this was overkill, considering voxel-rain was not far from our implementation of assignment 3. So we chose to scrap our original plan and refactored it into the following:

start Start Loads the canvas, and initializes G	initGL initShader
	createRenderProgram(fshader, vshader)
clearCanvas() clears the canvas by reinitializing GL	This class creates the program for rendering in GL
	(toRender) e or skull and sets the appropriate scale.
This object is what takes in the mesh ar	Renderer nd rendermethod and will draw the object. portion is stored here.
Dra	wable
var vertexBuffers var attributesEnabled var nElements var nAttributes var indexBuffer	
void draw(attribLocations)	
getBounds() draw()	

drawSelectedObject

This contains the logic for creating multiple renderables spawning them, moving them, and calling the appropriate draw with our camera translation

Camera
var center var diagonal var name
var at var eye var up var near var far var FOV
var getRotatedCameraPosition(angle) var getViewMatrix(e) var getRotatedViewMatrix(angle) var getProjMatrix() var tick() /* animates camera */

The graphics portion of this project lies entirely in the ModelRenderer. It takes in the appropriate data and draws itself where it needs to be. The model renderer is almost identical to the one found in assignment 3, aside from a few minor changes in order to get wire frames also drawing.

The Algorithm portion all lies within drawSelectedObject, where the multiple objects are spawned, the camera is moved, and the objects fall. It creates an array of objects, which have their locations controlled by a simple update code. Then the camera draws an object at that location.

Work Division

Each member contributed to code in some way or another, but aside from code, this is the main contributions that each person made

Dan Gau: Original Design, Implementation Plan and Scraped code. Adam Tango: Set up a Github Repo, Got the original design rolling and organized the team Jeffrey Klarfeld: Managed Github repo, provided base project code, finished final product.

Ed Jones: Made it rain; got our falling objects to run and made scalable code for tweaking Bradley Streu: Plan and code references in addition to communication facilitation