Lab Accident Simulator in Virtual Reality

Summer 2017

By

Opin Patel

Computer Science

University of Illinois at Chicago

Faculty Advisor:

Prof. John Bell

Content

Introduction:

Every year there are many accidents in laboratories and many of them can be avoided by properly following the safety rules. Even when lab participants feel safe they underestimate the situation and their surroundings thinking they know what they are doing and don't care about other unexpected events that may happen.

It is very difficult to teach someone what happens when there is a serious burn on eyes for example in real life, but we can give them the lesson in virtual world without harming them physically. They will remember the incident every time they enter the real lab. It will give them the importance of safety.

Purpose:

The purpose of this undergraduate research is to create virtual laboratory accident, which awe the user while he is immersed in virtual world, so as to produce a lasting memory of the consequences of disregarding lab safety, and thereby promote improved lab safety in the future. The part of this research goal is to create a better virtual world using two completely different platforms and compare them side by side. Also, these projects will be available on the VRUPL UIC website for anyone to download and run the simulation. This document will also include sites where the user can learn more about safety and about lab accidents. This document will also include step by step process to follow on how to work with different platforms.

Accident:

Unity:

The player will start off with a small room surrounded by white walls. Using the HTC VIVE controllers, he/she can teleport around the room using hair trigger on the back of the controller. Participant will have choice to wear safety goggles or not. To wear safety googles, all he needs to do is come close and stare at the goggles. He will see nice animation with the view divided in two eyes. Now, walking toward and looking at the door will animate and open up welcoming participants to enter the lab.

Now, the scenario is that while he is checking out the lab and walking through it, he will be tempted to check out the chemical reactor at the end of the lab, right before the periodic table. As he gets close to it, the reactor will spray orange chemical at him. If followed safety, hurray! He has avoided a dangerous accident and will survive the accident. If not, then he will hear a loud scream and his vision will be blocked by a video showing a blood lines coming down. He would not be able to see anything from that point on.

The main purpose of this accident is that participant must wear a proper safety googles in any chemistry lab. The video and audio in this demo will represent extreme content and injury on his eyes that can leave him blind, fortunately, for his entire VR life and teach him about safety.

I decided to do this way because eye is a precious part of the body and humans are more likely to remember an event if it involves visual and audible. This will create a long lasting memory.

OpenSceneGraph:

In OpenSceneGraph, the player will start out of lab. By using the mouse scroll button, he will be able to get close to the lab. Left clicking while dragging will rotate the view in desired direction. When he is close to the lab entrance, the massage will show up telling him what to do. By using keyboard, he will be able to pick up gas mask by “p” key and open door by using “o” key.

Just like Unity, the scenario will be bit similar. When the user gets close to the leaky chemical reactor, it will spray toxic chemical. This chemical will react with the air and turn it into a toxic dense smoke. If he is wearing a gas mask, participant will survive the toxic smoke and be able to escape the lab. If not, the participant will end up inhaling the gas and faint. Smoke will completely block his view leaving him helpless in the lab. Now, he will be in real trouble while waiting for the help to arrive while covered in toxic smoke.

Accidents in lab can turn dangerous due to many factors, for example, poor ventilation system (Students Faint in Corcoran Chemistry Lab.). Smoke in the air won’t get sucked out quickly, leaving participants in more trouble. This demo perfectly addresses that issue and that’s way it is important for participant to get VR experience and learn from it, remember for their life.

I decided to do this way because new participants in the lab are less aware of what they are surrounded by. Anything could go wrong at any time. Having a proper safety is the first step to avoid any accident. Hopefully, this VR experience will teach them good lifelong lesson.

Products Used:

Unity VR: Unity (unity3d) is a cross platform game engine developed by Unity Technologies, which is primarily used to develop video games and simulations for computers, consoles and mobile devices. Unity VR lets you target virtual reality devices directly from Unity, without any external plug-ins in projects. It provides a base API and feature set with compatibility for multiple devices. Unity is not available on Linux computers.

OpenSceneGraph: OpenSceneGraph (openscenegraph) is an open source high performance 3D graphics toolkit, used by application developers in fields such as visual simulation, games, virtual reality, scientific visualization and modelling. Written entirely in Standard C++ and OpenGL it runs on most platforms. OpenSceneGraph is now well established as the world leading scene graph technology and is widely used in many industries.

Product site: http://www.openscenegraph.org

Bash on Windows: Bash on Windows is a great way for windows users to get their work done in a Linux command line environment right within Microsoft Window, without worrying about creating a virtual machine. Bash on Windows runs Ubuntu user-mode binaries provided by Canonical. This means the command-line utilities are the same as those that run within a native Ubuntu environment. And the best thing is this feature is embedded inside Windows 10 anniversary update.

LunaPic: LunaPic (lunapic) is a great online simple tool that lets you upload an image from your computer and modify it. I mostly used LunaPic for cropping the images. Its Magic Wand feature would crop out the complex areas which is hard to do manually.
Product Website: http://www168.lunapic.com/editor/

Turbosquid: turbosquid (turbosquid)has a great collection of 3D models with Royalty Free License. Many models are in different file formats, which give you a freedom to choose the right one for the platform that you are working on. Product Website: https://www.turbosquid.com/

Xming X Server: Xming (Xming X Server for Windows) will support Graphical User Interface for the windows bash files. We will be using Xming to run the OpenSceneGraph applications on windows. Product Website: https://sourceforge.net/projects/xming/

SketchUp 2017: It was used to convert the file formats so that they are compatible with OpenSceneGraph and Unity (sketchup). By default it saves file with .skp file format, but model can be exported to many other formats like .3ds, .dxf, .dae, .fbx, .obj which is supported by Unity and OpenSceneGraph. Product Website: https://www.sketchup.com

Installation:

Installing bash shell on Windows:

Prerequisite: First, the computer needs to be running 64-bit version. Second, the computer needs to be running Windows 10 Anniversary Update. If your computer is running an older version, you can update it by going to Windows Setting, then Update & Security and then clicking on Check for Update. Now, we can install bash on Windows.

1.) On Windows open Setting -> click on Update & Security -> click on For developers -> select Developer mode option to set up bash environment.

Give permission to a warning pop up by clicking yes.

2.) Restart your computer after it installs the necessary components.

3.) After the computer reboots, open “Control Panel” -> click on “Programs” -> under “Programs and Features” click on “Turn Windows features on or off”.

4.) Check “Windows Subsystem for Linux (Beta)”.

5.) After the necessary installation, restart the computer.

6.) Now, from the search bar look for bash.exe and press Enter.

7.) On the command line, program will ask for your permission to move forward. Type “y” to agree.

8.) Create username and password when asked and let it install the necessary components. When done, close the window and try to open bash from search bar. That’s it! Now you are running bash on windows.

With bash you can access all of the files including windows and run them just like you would run files on standalone Unix command line.

Installing OpenSceneGraph on Windows bash:

Before we start, (Rudakova) let's install cmake, vim editor and git if not already installed. Open up bash command line window that we just installed and type the following commands:

~$ sudo apt-get install cmake

~$ sudo apt-get install vim

~$ sudo apt-get install git

Now, clone the latest version from OpenSceneGraph GitHub repository:

~$ git clone https://github.com/openscenegraph/OpenSceneGraph.git

Enter OpenSceneGraph Directory

~$ cd OpenSceneGraph

Install OpenGL utility toolkit

~/OpenSceneGraph $ sudo apt install freeglut3-dev

Now, Inside OpenSceneGraph runs the installation:

~/OpenSceneGraph $ mkdir build

~/OpenSceneGraph $ cd build

~/OpenSceneGraph/build $ cmake ..

Since we already installed freeglut3-dev files, it should not give any errors. If you get any extra errors, it would most likely that you need to install more components. Go through the commands and try to find any missing libraries and download it with command "sudo apt install <fileName>.

~/OpenSceneGraph/build $ make

~/OpenSceneGraph /build $ sudo make install

It might take about an hour to install.

Next, we will set environment variables that are used by OSG. To add it, we will use vim editor or your choice of any editor.

~/OpenSceneGraph $ vim ~/.bashrc

Once you are inside .bashrc file, press "i" to edit the document. Copy and paste these four lines at the end of the file.

```
export LD_LIBRARY_PATH="/usr/local/lib64:/usr/local/lib:$LD_LIBRARY_PATH"
```

export OPENTHREADS_INC_DIR="/usr/local/include"

export OPENTHREADS_LIB_DIR="/usr/local/lib64:/usr/local/lib"

export PATH="$OPENTHREADS_LIB_DIR:$PATH"

When you are done “:x” to save and exit the document and go back to the command screen.

Installing Xming:

Now close the bash window. In order for the GUI to work we need to download Xming on windows.

Visit https://sourceforge.net/projects/xming to download and set up on Windows.

Now, we need to connect Xming to Ubuntu bash, so open up a bash shell and type: export DISPLAY=localhost:0.0

Now, it’s fun time to download OSG examples and run it. By using bash command window create a folder named “data”.

~/usr/local/OpenSceneGraph$ mkdir data

Now enter the “data” folder and download the osg-data using git command:

~/usr/local/OpenSceneGraph/data$ git clone https://github.com/openscenegraph/OpenSceneGraph-Data.git

Now, add a data path as an environment variable to .bashrc file again.

export OSG_FILE_PATH="/usr/local/OpenSceneGraph/data/osg-data"

Now, you can run command “osgversion” or “osgviewer cow.osg” on bash command line window.

Upon this error “A Unable to open display ":0.0" Viewer::realize() - failed to set up any windows”, start your Xming server by clicking on the Xming icon.

Enabling VR in Unity:

Setting up Unity on windows computer is very easy. To create new project start Unity then on New and then give name and choose 3D option then Create project.

After creating a project, visit Edit > Project Settings> Player > on right side of window expand Other Settings > check “Virtual Reality Supported”.

Enabling HTC Vive for Unity:

Inside Unity, click on the Asset Store and find SteamVR Plugin. Install SteamVR Plugin and then import all the assets.

Now, open File->Build Settings->Player Settings. On the right hand side expand other Settings. Under Virtual Reality SDKs click on '+' sign. Now, add OpenVR to it.

Now, click on Project tab and navigate to Assets->SteamVR->Prefabs. Drag and drop CameraRig and SteamVR under the project hierarchy.

Now, we need to unable laser pointer and Teleportation using Vive controllers, so we can render the scene.

Visit Assets->SteamVR->Extras folder drag files SteamVR_LaserPointer and SteamVR_Teleporter and drop them to the project hierarchy CameraRig->Controller (Left) and to Controller (Right).

Now, you should be able to move around your scene with HTC Vive controllers.

Development and Production:

Unity:

Unity is well known for its well-developed platform that makes it very easy for anyone to create a 3d virtual world. It has support for almost all of the actions that we encounter in real life and provides an entire assets store dedicated to Unity developers. It supports major VR devices and has built in libraries for that.

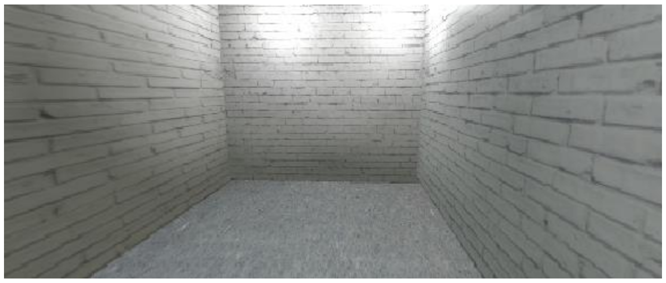


Figure 1

- To create a physical model in Unity, first you need to create a 3D game object.
- In this case, I am using cube and stretching and positioning it to my need.
- Then you can upload the 2D image of your choice.
- Attach an image to the cube, it will wrap around the cube creating a wall like texture.
- Also you can drop 3D models to it and position them to your need.

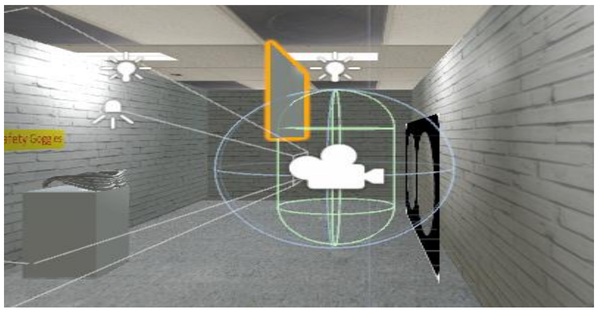


Figure 2

- Without camera, there is no point to work with VR.
- Simple camera will sit at one place with the fixed views; I have applied many components like Simple Mouse Rotator, First Person Controller, so the user can walk around in VR with keyboard input and look around using mouse movement.
- Also, I have attached 3D game Objects that would follow the main camera.
- Also, it is good idea to attach Physics RayCast for interaction in VR.

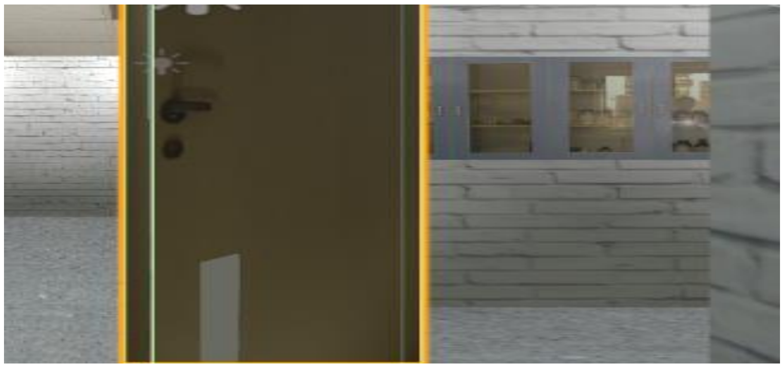


Figure 3

- Animation is very important to create a real life scenario in VR.
- When triggered, this door will smoothly open up welcoming you to enter the laboratory, other animations are wearing a safety goggles, chemical reactor spraying chemical all around the room, dropping a screen right before the main camera making the person bleed which is the video attached to the quad.
- Animations can be created within the Unity and needs to be scripted on when to trigger.

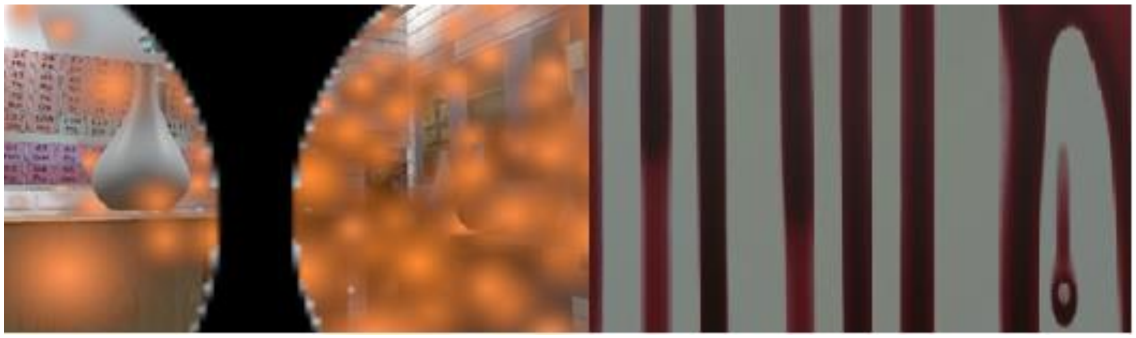


Figure 4

- Here is the example of an interaction, when the player gets close to the chemical reactor, as he is not aware, it will spray chemical on the person.
- Left image shows an unexpected accident that happened in lab while the person was protected with his safety goggles, so it would not cause any harm to him.
- Right image is the example of improper safety, upon accident the person would lose his view of the lab completely and the blood would come down from his eyes. It is shown by the video effect.
- We are also planning the audible scream sound when the accident happens.

The best way to learn about the features of Unity is through the Unity Documentation page. There are also many YouTube tutorials that would also help.

OpenSceneGraph:

OpenSceneGraph (Wang and Xuelei)supports multiple platforms making it easy to share your work. Starting a development is simple; all you need is file editor and software to run the 3D program. Unlike Unity, everything in OpenSceneGraph is code based, which gives you more customizability. It requires good amount of learning the platform before you can start making your model .

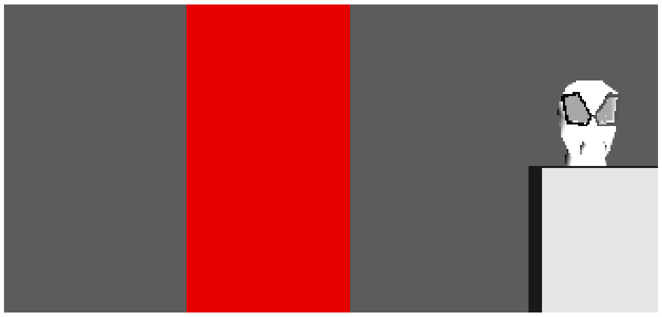


Figure 5

- For this platform, I decided to keep it simple to speed up the process of learning plus implementing.
- Here, I have a red door and a 3D model outside the lab.
- OpenSceneGraph has Geometry class which can be useful to create different geometry shapes and you can assign color to it.
- Also, it supports many 3D files format, which is very easy to upload and place it.

- When it comes to handling camera, OSG (Kapelko) provides many manipulations that are widely used in industries:
    - DriveManupulator, which is similar to driving a car using the up, down arrow to accelerate/deaccelerate and space bar to stop.
    - FlightManipulator to handle the camera view from cockpit.
    - UFOManipulator which is closest to a FirstPersonShooter manipulator. You can go up, down, forward, backward, look around using keyboard arrow inputs.
    - I decided to use TrackballManipulator, which is very simple compare to other options. Mouse scroll bar to go back and forth, left click drag to look around.

Figure 6

- OSG supports interaction via mouse and keyboard events.
- For this program, I will pick up face mask using “p” and open door using “o” key
- Each keyboard key can be assigned for particular action.
- Also, you can create invisible force field area around an object so when you get close to it, it triggers an action.

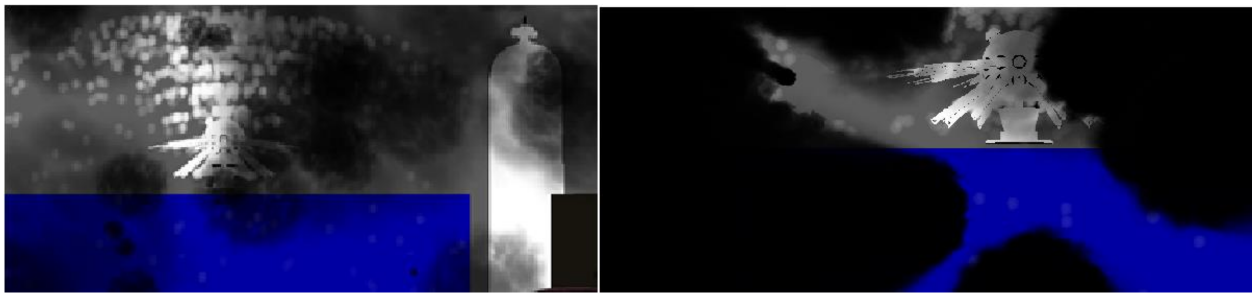


Figure 7

- With the safety mask on the face in Left image, person would be able to survive the situation even with the dense smoke.
- On the right image, without the mask his vision will fade out and eventually he would not be able to see anything except the black dark smoke.
- Program will also include audio script which will tell him his feature with the severe damage to his body.
- Visual plus audio will create a long term memory in participant the importance of safety in the lab.

The best way to learn about OpenSceneGraph is through their Quick Start Guide and using the book OpenSceneGraph 3.0: Beginner's Guide.

Comparison Between Two Software Products:

The second purpose of this project was to compare the two products Unity and OpensceneGraph on many different aspects and evaluate them by giving points. In the following table two products are rated from 1 to 10 in a number of key criteria.

Evaluation Criteria	OpenSceneGraph	Unity
1. Installation	3	10
2. Platform Support	10	5
3. Cross Platform portability	10	5
4. Simplicity	9	5
5. Usability for perspective audience	5	9
6. Output Performance	7	10
7. Scalability	10	10
8. Optimization	10	10
9. Size of file	10	5
10. Help/Support	7	9
11. Support for VR devices	5	10
Total - 110	86	88

When it comes to a development of 3D model, both Unity and OSG would be able to create a great model that you are looking for. Unity is a great tool for someone with little to no knowledge of code, whereas OpenSceneGraph requires good understanding of Object oriented language. Also, Unity has a nice user interface which makes development even easier. In Comparison, OSG is more customizable. Unity includes many assets, libraries and VR enabled devices. One of the biggest drawback is it is not supported on Linux environment.

Future work:

- These projects can be further expanded to have a multiple rooms with different accidents. Depending on the environment different people work. For example, we can have biology lab, chemistry lab, industrial lab which gives them the feel for their environment and involving them in accidents that are most common to their field.
- Also, the user interface can still be improved with more functionality and objects. It would be great if it looks just like actual lab and participant has more control over the work or activity he is doing.
- If technology becomes available, it is great to have group of people being in the same lab at the same time witness one of their member with improper safety gets knocked out by an accident. This will create a conversation among them about safety.

Conclusion:

Now that we have great tools like Unity and OpenSceneGraph, it can be greatly used to improve the life of people by immersing them in the virtual world and teaching great things. For this project, I was successfully able to create models in two different platforms. The impact of the virtual accident on the user will certainly create awareness toward safety. The comparison between two platforms will certainly help the developer to make right choice.

Works Cited

1. DeCarlo, Paul. "channel9.msdn.com." 11 04 2016. *https://channel9.msdn.com/Blogs/WinCoder/XMING--Bash-on-Ubuntu-on-Windows--X11-Window-System-Running-from-Windows10-Subsystem-for-Linux.* 01 07 2017.
2. *https://3dwarehouse.sketchup.com/*. n.d. 20 July 2017.
3. *https://www.turbosquid.com/*. n.d. Website. 25 July 2017.
4. Kapelko, Michae. *Install OpenSceneGraph under Linux. OpenSceneGraph Cross-platform Guide 01*. 08 03 2017. YouTube.
5. *lunapic*. n.d. 20 July 2017. <http://www168.lunapic.com/editor/>.
6. Martz, Paul. *OpenSceneGraph Quick Start Guide: a Quick Introduction to the Cross-Platform Open Source Scene Graph API*. Skew Matrix Software, 2007. Print.
7. *openscenegraph*. n.d. 25 July 2017. <www.openscenegraph.com>.
8. Rudakova, Victoria. "https://vicrucann.github.io/tutorials/osg-linux-quick-install/." 06 12 2015. *github.io.* 03 08 2017.
9. *sketchup*. n.d. <www.sketchup.com/products/sketchup-pro/new-in-2017.>.
10. "Students Faint in Corcoran Chemistry Lab." 29 April 2007. *The GW Hatchet.* Web.
11. "Support/UserGuides/Examples – Osg." n.d. *openscenegraph.org.* 03 08 2017.
12. *turbosquid*. n.d. 25 July 2017. <https://www.turbosquid.com/>.
13. *unity3d*. n.d. Website. 25 July 2017. <https://unity3d.com/learn/tutorials/topics/virtual-reality>.
14. Wang, Rui and Qian Xuelei. *OpenSceneGraph 3.0: Beginner's Guide; Create High-performance Virtual Reality Applications with OpenSceneGraph, One of the Best 3D Graphics Engines*. Packt Publishing, 2010. Print.
15. *Welcome to the Unity Scripting Reference!* n.d. Web. 25 July 2017.
16. *Xming X Server for Windows*. n.d. <sourceforge.net/projects/xming/>.

Images:

Unity:

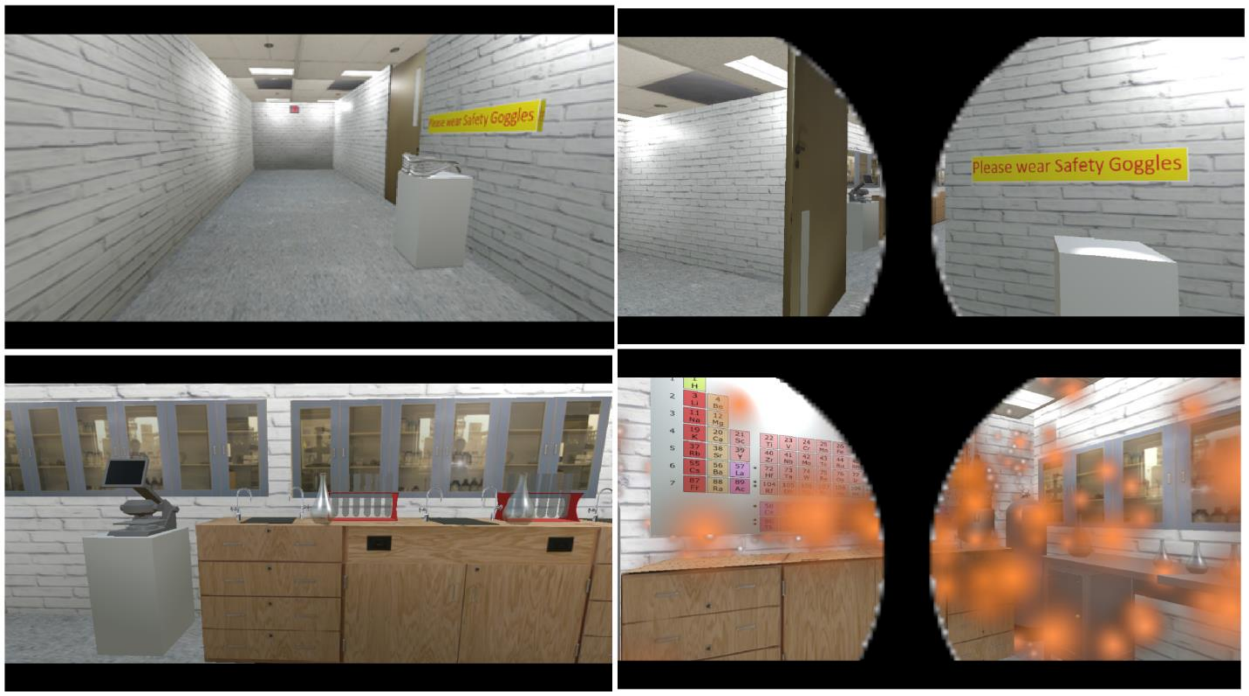


Figure 8

Left images show the lab without the goggles while the right ones show the door animation and accident in effect

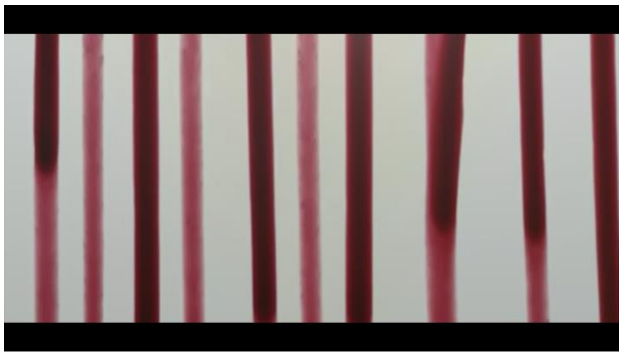
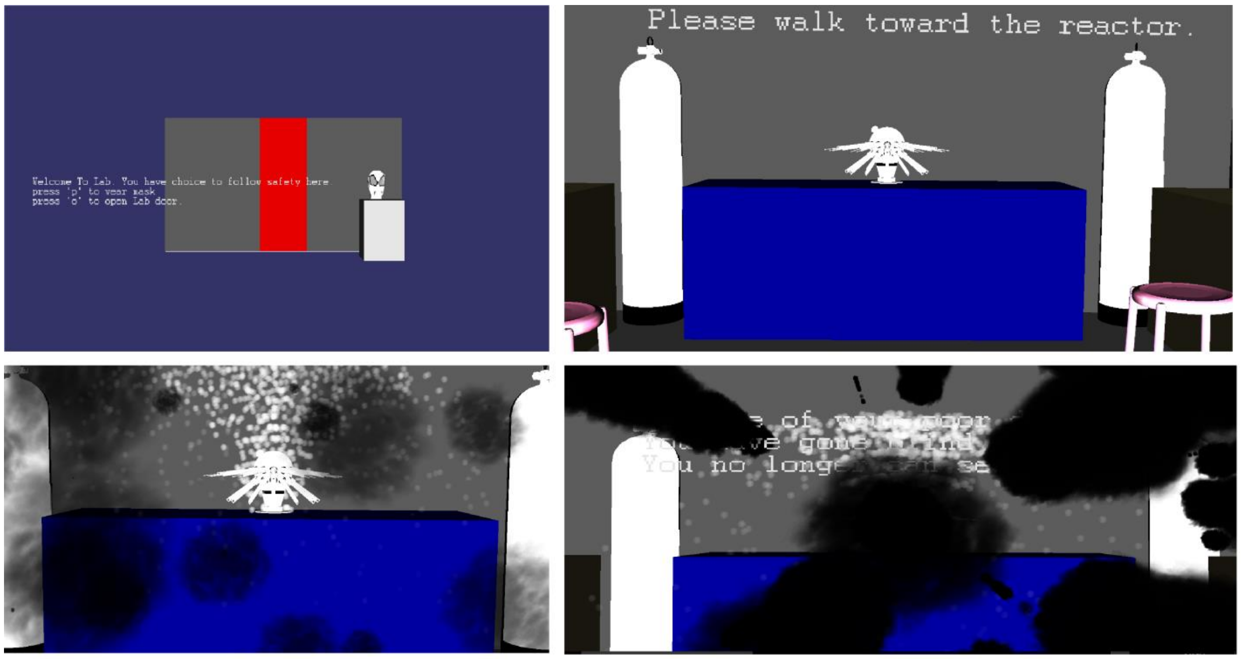


Figure 9

Blood lines coming down after the accident blocking the camera view

OpenSceneGraph:



```
Welcome To Lab. You have choice to follow safety here.
press 'p' to wear mask
press 'o' to open Lab door.
```

```
Please walk toward the reactor.
```

```
e of [illegible]
[illegible] ve gone [illegible]
You no longe [illegible] se
```

Figure 10

Top images show the lab model, bottom left shows accident with mask on and bottom right without the mask will cover the entire user view.

Code:

Unity:

```
/**
 * File name: rayCastObj.cs
 * script includes raycast and its interaction with objects
 * By: Opin Patel
 * UIC CS 398 Undergraduate Research
 *
**/

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class rayCastObj : MonoBehaviour { // rayCastObj class
    bool doorStatus = false;
    bool glassessOn = false;
    public GameObject explosion;
    public GameObject fire;
    public string disText;
    public Transform target;
    public GameObject onFace;
    public GameObject eyeBleedObject;
    public GameObject spray1ani;
    bool spray1Played = false;
    bool showText = false;
    bool screamPlayed = false;
    public GameObject screamObj;

    // Update is called once per frame
    void Update () {
        RaycastHit hit;      // raycast hit will detect the lazer collision with objects.
        float theDistance;
        GameObject dDoor;

        // forward vector that will be used with RaycastHit
        Vector3 forward = transform.TransformDirection (Vector3.forward) * 1;
        Debug.DrawRay (transform.position, forward, Color.green);

        // if the lazer hit the door, then we want it to animate using doorOpen animation.
        if (Physics.Raycast (transform.position, (forward), out hit, 4)) {
          if (hit.collider.gameObject.tag == "door" && doorStatus == false) {
             hit.collider.gameObject.GetComponent<Animation> ().Play ("doorOpen");
             doorStatus = true;
          }
       }

       // same if it hit the safety googles, then play the glassessOnFace animation.
       if (Physics.Raycast (transform.position, (forward), out hit, 4)) {
          if (hit.collider.gameObject.tag == "tableGlass") {
             Destroy (hit.collider.gameObject);
             //print ("wearing glassess");
             onFace.GetComponent<Animation> ().Play ("glassessOnFace");
             glassessOn = true;
          }

          if (hit.collider.gameObject.tag == "chemicalReactor" || hit.collider.gameObject.tag == "ta
bleExplo" || hit.collider.gameObject.tag == "flask") {
             // Here, we want the explosition particle system to go off.
             Instantiate (explosion, transform.position, transform.rotation);
             if (glassessOn == false) {
                eyeBleedObject.GetComponent<Animation> ().Play ("newAnimation");
                screamObj.GetComponent<AudioSource> ().Play ();
             }
             if (glassessOn == true && spray1Played == false) {
                print ("spraying animation");
                spray1ani.GetComponent<Animation> ().Play ("spot1new");
                spray1Played = true;
             }
          }
       }
    }
}
```

OpenSceneGraph:

```
/**
 * File name: 3dmodel.cpp
 * file includes code for the OSG model
 * By: Opin Patel
 * UIC CS 398 Undergraduate Research
 *
**/

#include <iostream>
#include <osg/ShapeDrawable>
#include <osg/Geode>
#include <osg/Geometry>
#include <osg/Group>
#include <osg/MatrixTransform>
#include <osg/Camera>
#include <osg/Point>
#include <osg/PointSprite>
#include <osg/Texture2D>
#include <osg/BlendFunc>
#include <osg/Material>
#include <osg/StateSet>
//#include <osgGA/FlightManipulator>
#include <osgGA/TrackballManipulator>
//#include <osgGA/TerrainManipulator> // different camera manipulators
//#include <osgGA/DriveManipulator>
//#include <osgGA/UFOManipulator>
#include <osgGA/GUIEventHandler>
#include <osgGA/CameraManipulator>
#include <osgGA/StateSetManipulator>
#include <osgParticle/ParticleSystem>
#include <osgParticle/ParticleSystemUpdater>
#include <osgParticle/ModularEmitter>
#include <osgParticle/ModularProgram>
#include <osgParticle/AccelOperator>
#include <osgText/Text>
#include <osgText/Font>
#include <osgViewer/Viewer>
#include <osgDB/ReadFile>
#include <osgViewer/ViewerEventHandlers>
```

```
using namespace osg; //using osg namespace

bool gogglesOn = false;
bool once = false;

// ModelController will inherit GUIeventhandler class for keyboard input events
class ModelController : public osgGA::GUIEventHandler
{
        public:
        ModelController( osg::MatrixTransform* node , osg::MatrixTransform* goggle)
        : _model(node), _goggle (goggle)
        {}
        virtual bool handle( const osgGA::GUIEventAdapter& ea,osgGA::GUIActionAdapter& aa
);
        protected:
        osg::ref_ptr<osg::MatrixTransform> _model;
        osg::ref_ptr<osg::MatrixTransform> _goggle;
};

// plays different actions on keyboard events
bool ModelController::handle( const osgGA::GUIEventAdapter& ea, osgGA::GUIActionAdapter&
aa )
{
        if ( !_model ) return false;
        if ( !_goggle ) return false;
        osg::Matrix matrix = _model->getMatrix();
        osg::Matrix goggleMatrix = _goggle->getMatrix();

        switch ( ea.getEventType() )
        {
                case osgGA::GUIEventAdapter::KEYDOWN:
                switch ( ea.getKey() )
                {
                        case 'o': case 'O':
                        matrix *= osg::Matrix::rotate(-0.5f, osg::Z_AXIS);  // will open the door
                        break;
                        case 'c': case 'C':
                        matrix *= osg::Matrix::rotate(0.5f, osg::Z_AXIS);  // will close the door
                        break;
                        case 'p': case 'P':
```

```
				goggleMatrix *= osg::Matrix::scale(0.0f,0.0f,0.0f); // put mask on the face
				gogglesOn = true;
				break;

				default:
				break;
			}

	_model->setMatrix( matrix );
	_goggle->setMatrix( goggleMatrix );
	break;
	default:
	break;
	}
return false;
}

//particle spray
osgParticle::ParticleSystem* createParticleSystem(osg::Group* parent )
{
	//creates a new particle system and sets its shape
	osg::ref_ptr<osgParticle::ParticleSystem> ps = new osgParticle::ParticleSystem;
	ps->getDefaultParticleTemplate().setShape(osgParticle::Particle::POINT );

	osg::ref_ptr<osg::BlendFunc> blendFunc = new osg::BlendFunc;
	blendFunc->setFunction( GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA );
	osg::ref_ptr<osg::Texture2D> texture = new osg::Texture2D;
	//loading the 2d texture using the rbg iamge
	texture->setImage( osgDB::readImageFile("smoke.rgb"));

	osg::StateSet* ss = ps->getOrCreateStateSet();
	ss->setAttributeAndModes( blendFunc.get() );
	ss->setTextureAttributeAndModes( 0, texture.get() );
	ss->setAttribute( new osg::Point(20.0f) );
	ss->setTextureAttributeAndModes( 0, new osg::PointSprite );
	ss->setMode( GL_LIGHTING, osg::StateAttribute::OFF);
	ss->setRenderingHint( osg::StateSet::TRANSPARENT_BIN );

	// can set the rate at which the particle shoots
	osg::ref_ptr<osgParticle::RandomRateCounter> rrc = new
osgParticle::RandomRateCounter;
```

```
        rrc->setRateRange( 500, 800 );

        osg::ref_ptr<osgParticle::ModularEmitter> emitter = new osgParticle::ModularEmitter;
        emitter->setParticleSystem( ps.get() );
        emitter->setCounter( rrc.get() );

        osg::ref_ptr<osgParticle::AccelOperator> accel = new osgParticle::AccelOperator;
        accel->setToGravity();

        osg::ref_ptr<osgParticle::ModularProgram> program = new
osgParticle::ModularProgram;
        program->setParticleSystem( ps.get() );
        program->addOperator( accel.get() );

        osg::ref_ptr<osg::Geode> geode = new osg::Geode;
        geode->addDrawable( ps.get() );
        parent->addChild( emitter.get() );
        parent->addChild( program.get() );
        parent->addChild( geode.get() );
        return ps.get();
}

int main(int argc, char** argv)
{
        // creating a shapeDrawable object and assigning size and position and finally the color
        osg::ref_ptr<osg::ShapeDrawable> leftWall = new osg::ShapeDrawable;
        leftWall->setShape(new osg::Box(osg::Vec3(-10.0f, 0.0f, 0.0f), 0.1f, 20.0f, 10.0f));
        leftWall->setColor(osg::Vec4(0.4f, 0.4f, 0.4f, 0.0f));   //red, green, blue, black

        osg::ref_ptr<osg::ShapeDrawable> rightWall = new osg::ShapeDrawable;
        rightWall->setShape(new osg::Box(osg::Vec3(10.0f, 0.0f, 0.0f), 0.1f, 20.0f, 10.0f));
        rightWall->setColor(osg::Vec4(0.4f, 0.4f, 0.4f, 0.0f));

        osg::ref_ptr<osg::ShapeDrawable> backWall = new osg::ShapeDrawable;
        backWall->setShape(new osg::Box(osg::Vec3(0.0f, 10.0f, 0.0f), 20.0f, 0.1f, 10.0f));
        backWall->setColor(osg::Vec4(0.4f, 0.4f, 0.4f, 0.0f));

        osg::ref_ptr<osg::ShapeDrawable> frontWallLeft = new osg::ShapeDrawable;
        frontWallLeft->setShape(new osg::Box(osg::Vec3(-6.0f, -10.0f, 0.0f), 8.0f, 0.1f, 10.0f));
        frontWallLeft->setColor(osg::Vec4(0.4f, 0.4f, 0.4f, 0.0f));
```

```
	osg::ref_ptr<osg::ShapeDrawable> frontWallRight = new osg::ShapeDrawable;
	frontWallRight->setShape(new osg::Box(osg::Vec3(6.0f, -10.0f, 0.0f), 8.0f, 0.1f, 10.0f));
	frontWallRight->setColor(osg::Vec4(0.4f, 0.4f, 0.4f, 0.0f));

	osg::ref_ptr<osg::ShapeDrawable> door = new osg::ShapeDrawable;
	door->setShape(new osg::Box(osg::Vec3(0.0f, -10.0f, 0.0f), 4.0f, 0.1f, 10.0f));
	door->setColor(osg::Vec4(1.0f, 0.0f, 0.0f, 0.0f));

	osg::ref_ptr<osg::ShapeDrawable> floor = new osg::ShapeDrawable;
	floor->setShape(new osg::Box(osg::Vec3(0.0f, 0.0f, -5.0f), 20.0f, 20.0f, 0.1f));

	osg::ref_ptr<osg::ShapeDrawable> tableLeft = new osg::ShapeDrawable;
	tableLeft->setShape(new osg::Box(osg::Vec3(-8.5f, 0.0f, -3.0f), 3.0f, 16.0f, 4.0f)); //lenth,
width, height
	tableLeft->setColor(osg::Vec4(0.9f, 0.8f, 0.5f, 0.0f));

	osg::ref_ptr<osg::ShapeDrawable> tableRight = new osg::ShapeDrawable;
	tableRight->setShape(new osg::Box(osg::Vec3(8.5f, 0.0f, -3.0f), 3.0f, 16.0f, 4.0f));
//lenth, width, height
	tableRight->setColor(osg::Vec4(0.9f, 0.8f, 0.5f, 0.0f));

	osg::ref_ptr<osg::ShapeDrawable> tableBack = new osg::ShapeDrawable;
	tableBack->setShape(new osg::Box(osg::Vec3(0.0f, 8.5f, -3.0f), 10.0f, 3.0f, 4.0f));
	tableBack->setColor(osg::Vec4(0.0f, 0.0f, 0.7f, 0.0f));

	osg::ref_ptr<osg::ShapeDrawable> tableOut = new osg::ShapeDrawable;
	tableOut->setShape(new osg::Box(osg::Vec3(7.5f, -15.0f, -3.0f), 3.0f, 3.0f, 4.0f)); //lenth,
width, height

	//left tubes objects
	osg::ref_ptr<osg::ShapeDrawable> tube1 = new osg::ShapeDrawable;
	tube1->setShape(new osg::Cylinder(osg::Vec3(-8.5f, -1.0f, -0.5f), 0.1f, 1.0f));
	osg::ref_ptr<osg::ShapeDrawable> tube2 = new osg::ShapeDrawable;
	tube2->setShape(new osg::Cylinder(osg::Vec3(-8.5f, -0.5f, -0.5f), 0.1f, 1.0f));
	osg::ref_ptr<osg::ShapeDrawable> tube3 = new osg::ShapeDrawable;
	tube3->setShape(new osg::Cylinder(osg::Vec3(-8.5f, 0.0f, -0.5f), 0.1f, 1.0f));
	osg::ref_ptr<osg::ShapeDrawable> tube4 = new osg::ShapeDrawable;
	tube4->setShape(new osg::Cylinder(osg::Vec3(-8.5f, 0.5f, -0.5f), 0.1f, 1.0f));

	//right tubes
	osg::ref_ptr<osg::ShapeDrawable> tube5 = new osg::ShapeDrawable;
	tube5->setShape(new osg::Cylinder(osg::Vec3(8.5f, -3.0f, -0.5f), 0.1f, 1.0f));
	osg::ref_ptr<osg::ShapeDrawable> tube6 = new osg::ShapeDrawable;
	tube6->setShape(new osg::Cylinder(osg::Vec3(8.5f, -2.5f, -0.5f), 0.1f, 1.0f));
	osg::ref_ptr<osg::ShapeDrawable> tube7 = new osg::ShapeDrawable;
	tube7->setShape(new osg::Cylinder(osg::Vec3(8.5f, -2.0f, -0.5f), 0.1f, 1.0f));
	osg::ref_ptr<osg::ShapeDrawable> tube8 = new osg::ShapeDrawable;
	tube8->setShape(new osg::Cylinder(osg::Vec3(8.5f, -1.5f, -0.5f), 0.1f, 1.0f));

	//loading 3D model files and applying manipulations to it
	osg::ref_ptr<osg::Node> fire = osgDB::readNodeFile("SmokeBox.osgt");
	osg::ref_ptr<osg::Node> stool = osgDB::readNodeFile("chl009.3DS");
	osg::ref_ptr<osg::Node> cylinder = osgDB::readNodeFile("gasCylinder.3ds");

	// to resize or position 3d model, matrix scale and translate can be used
	osg::ref_ptr<osg::MatrixTransform> cylinderTransformLeft = new osg::MatrixTransform;
	osg::Matrix cylinderPosLeft = osg::Matrix::translate(-2000.0f, 1900.0f, -1300.0f);
	osg::Matrix cylinderScaleLeft = osg::Matrix::scale(0.004f,0.004f,0.004f);
	cylinderTransformLeft->setMatrix(cylinderPosLeft * cylinderScaleLeft);
	cylinderTransformLeft->addChild(cylinder);

	osg::ref_ptr<osg::MatrixTransform> cylinderTransformRight = new
osg::MatrixTransform;
	osg::Matrix cylinderPosRight = osg::Matrix::translate(1300.0f, 1900.0f, -1300.0f);
	osg::Matrix cylinderScaleRight = osg::Matrix::scale(0.004f,0.004f,0.004f);
	cylinderTransformRight->setMatrix(cylinderPosRight * cylinderScaleRight);
	cylinderTransformRight->addChild(cylinder);


	osg::ref_ptr<osg::Node> chemReactor = osgDB::readNodeFile("chemDevice.3ds");
	osg::ref_ptr<osg::MatrixTransform> chemTransform = new osg::MatrixTransform;
	osg::Matrix chemPos = osg::Matrix::translate(-10.0f, 70.0f, -10.0f);
	osg::Matrix chemScale = osg::Matrix::scale(0.09f,0.09f,0.09f);
	chemTransform->setMatrix(chemPos * chemScale);
	chemTransform->addChild(chemReactor);

	osg::ref_ptr<osg::MatrixTransform> stoolTransform = new osg::MatrixTransform;
	osg::Matrix posStool = osg::Matrix::translate(90.0f, 30.0f, -85.0f);		// right side
back
	osg::Matrix sizeStool = osg::Matrix::scale(0.06f, 0.06f, 0.06f);
	osg::Matrix resultMatStool = posStool * sizeStool;
	stoolTransform->setMatrix( resultMatStool );
	stoolTransform->addChild(stool.get());

	osg::ref_ptr<osg::MatrixTransform> stoolTransformFront = new osg::MatrixTransform;
	osg::Matrix posStoolFront = osg::Matrix::translate(90.0f, -60.0f, -85.0f);		// right
side front
	osg::Matrix sizeStoolFront = osg::Matrix::scale(0.06f, 0.06f, 0.06f);
	osg::Matrix resultMatStoolFront = posStoolFront * sizeStoolFront;
	stoolTransformFront->setMatrix( resultMatStoolFront);
	stoolTransformFront->addChild( stool.get() );

	osg::ref_ptr<osg::MatrixTransform> stoolTransformLeft = new osg::MatrixTransform;
	osg::Matrix posStoolLeft = osg::Matrix::translate(-90.0f, 0.0f, -85.0f);		// leftSide
	osg::Matrix sizeStoolLeft = osg::Matrix::scale(0.06f, 0.06f, 0.06f);
	osg::Matrix resultMatStoolLeft = posStoolLeft * sizeStoolLeft;
	stoolTransformLeft->setMatrix( resultMatStoolLeft);
	stoolTransformLeft->addChild( stool.get() );

	osg::ref_ptr<osg::Node> glassNode = osgDB::readNodeFile("gasMask.obj");
	osg::ref_ptr<osg::MatrixTransform> transform1 = new osg::MatrixTransform;
	osg::Matrix pos = osg::Matrix::translate(1500.0f, 750.0f, -250.0f);
	osg::Matrix rotate = osg::Matrix::rotate(1.5f, Vec3(0.0f, 0.0f, -600.0f));
	osg::Matrix size = osg::Matrix::scale(0.01f, 0.01f, 0.01f);
	osg::Matrix resultMat = pos * size * rotate;
	transform1->setMatrix( resultMat );
	transform1->addChild( glassNode.get() );

	osg::ref_ptr<osg::Node> tubeHolder = osgDB::readNodeFile("tubeholder.dae");
	osg::ref_ptr<osg::MatrixTransform> tubeHolderTrans = new osg::MatrixTransform;
	osg::Matrix tubeHolderpos = osg::Matrix::translate(0.0f, 0.0f, 0.0f);
	osg::Matrix tubeHoldersize = osg::Matrix::scale(1000.0f, 1000.0f, 1000.0f);
	osg::Matrix tubeHolderresultMat = tubeHolderpos * tubeHoldersize;
	tubeHolderTrans->setMatrix( tubeHolderresultMat );
	tubeHolderTrans->addChild( tubeHolder.get());


	//adding texts to file
	osg::ref_ptr<osgText::Font> g_font = osgText::readFontFile("arial.ttf");

	osg::ref_ptr< osgText::Text > textLabInfo = new osgText::Text;
	textLabInfo->setText( "Welcome To Lab. You have choice to follow safety here.\npress
'p' to wear mask\npress 'o' to open Lab door." );
```

```
        textLabInfo->setFont( g_font.get() );
        textLabInfo->setCharacterSize( 0.5f );
        textLabInfo->setAxisAlignment( osgText::TextBase::XZ_PLANE );
        osg::ref_ptr<osg::Geode> textGeode = new osg::Geode;
        textGeode->addDrawable(textLabInfo.get());

        osg::ref_ptr< osgText::Text > textInLab = new osgText::Text;
        textInLab->setText( "Please walk toward the reactor." );
        textInLab->setFont( g_font.get() );
        //textInLab->setPosition( osg::Vec3( 0.0, 0.0, 0.0 ) );
        textInLab->setCharacterSize( 0.5f );
        textInLab->setAxisAlignment( osgText::TextBase::XZ_PLANE );
        osg::ref_ptr<osg::Geode> textInLabGeode = new osg::Geode;
        textInLabGeode->addDrawable(textInLab.get());

        osg::ref_ptr< osgText::Text > textBlind = new osgText::Text;
        textBlind->setText( "Because of your poor decision\nYou have gone blind\nYou no
longer can see anything" );
        textBlind->setFont( g_font.get() );
        //textBlind->setPosition( osg::Vec3( 15.0, -20.0, 0.0 ) );
        textBlind->setCharacterSize( 0.5f );
        textBlind->setAxisAlignment( osgText::TextBase::XZ_PLANE );
        osg::ref_ptr<osg::Geode> textBlindGeode = new osg::Geode;
        textBlindGeode->addDrawable(textBlind.get());

        //Door Animation : Start
        osg::ref_ptr<osg::Geode> doorGeode = new osg::Geode;
        doorGeode->addDrawable(door.get());

        osg::ref_ptr<osg::MatrixTransform> mt = new osg::MatrixTransform;
        mt->addChild( doorGeode.get() );
        osg::ref_ptr<ModelController> ctrler = new ModelController( mt.get(), transform1.get()
);
        //Door Animation : End

        //Spray : Start
        osg::ref_ptr<osg::MatrixTransform> sprayMt = new osg::MatrixTransform;
        sprayMt->setMatrix( osg::Matrix::translate(0.0f, 8.0f, -2.0f) );

        osgParticle::ParticleSystem* ps = createParticleSystem( sprayMt.get());
```

```
	osg::ref_ptr<osgParticle::ParticleSystemUpdater> updater = new
osgParticle::ParticleSystemUpdater;
	updater->addParticleSystem( ps );
	//Spray : End

	//creating tubeGeode for 8 tubes in the model then adding it to root node
	osg::ref_ptr<osg::Geode> tubeGeode = new osg::Geode;
	tubeGeode->addDrawable(tube1.get());
	tubeGeode->addDrawable(tube2.get());
	tubeGeode->addDrawable(tube3.get());
	tubeGeode->addDrawable(tube4.get());
	tubeGeode->addDrawable(tube5.get());
	tubeGeode->addDrawable(tube6.get());
	tubeGeode->addDrawable(tube7.get());
	tubeGeode->addDrawable(tube8.get());



	osg::ref_ptr<osg::Geode> root1 = new osg::Geode;
	root1->addDrawable(leftWall.get());
	root1->addDrawable(rightWall.get());
	root1->addDrawable(backWall.get());
	root1->addDrawable(frontWallLeft.get());
	root1->addDrawable(frontWallRight.get());
	root1->addDrawable(floor.get());
	root1->addDrawable(tableLeft.get());
	root1->addDrawable(tableRight.get());
	root1->addDrawable(tableBack.get());
	root1->addDrawable(tableOut.get());

	// root will contain everything that we have created so far and will be added to the
viewer
	osg::ref_ptr<osg::Group> root = new osg::Group;
	root->addChild(root1.get());
	root->addChild(tubeGeode.get());
	root->addChild( mt.get() );
	root->addChild(transform1.get() );
	root->addChild(stoolTransform.get());
	root->addChild(stoolTransformFront.get());
	root->addChild(stoolTransformLeft.get());
	root->addChild(chemTransform.get());
	root->addChild(cylinderTransformLeft.get());
	root->addChild(cylinderTransformRight.get());
	root->addChild(tubeHolderTrans.get());

	//starting a viewer and adding scene to it plus the event handler for keyboard input plus
trackball Manipulator
	osgViewer::Viewer viewer;
	// press o to open the door, c to close the door and p to pick up face mask
	viewer.addEventHandler( ctrler.get() );
	viewer.getCamera()->setAllowEventFocus( true );
	viewer.setSceneData(root);
	// left click+movement for angle, right click + movement to go forward backward,
middle click + movement to go sideways
	viewer.setCameraManipulator(new osgGA::TrackballManipulator);

	bool textdisappear1 = false;
	osg::ref_ptr<osg::MatrixTransform> textTrans = new osg::MatrixTransform;
	osg::ref_ptr<osg::MatrixTransform> textTrans1 = new osg::MatrixTransform;

	// until we terminate the program, while loop will keep updating the camera frames.
	// so, this is the best place to dynamically add or remove any child nodes from the root
while the program is on.
while ( !viewer.done() )
	{
		osg::Vec3 eye,center,up;
		viewer.getCamera()-> getViewMatrixAsLookAt(eye,center,up);
		//std::cout << center[0] <<"  "<< center[1] <<"  "<< center[2] << std::endl;
		viewer.frame();

		// when the player gets close to lab he will be presented with the text massage
on screen
		if (center[0] <= 4.0 && center[0] >= -4.0 && center[1] <= -30.0 && center[1] >= -
80.0 && text1 == false){
			osg::Matrix textpos = osg::Matrix::translate(center[0] - 15.0, center[1] +
50.0, center[2]);
			textTrans->setMatrix(textpos);
			textTrans->addChild(textGeode.get());
			root->addChild(textTrans.get());
			text1 = true;
		}
		// now when he gets even close we will the text from the screen
		if (center[0] <= 4.0 && center[0] >= -4.0 && center[1] <= -10.0 && center[1] >= -
40.0 && text1 == true){
			root->removeChild(textTrans.get());
		}

		if (center[0] <= 4.0 && center[0] >= -4.0 && center[1] <= -1.0 && center[1] >= -
10.0 && text2 == false){
			osg::Matrix textpos1 = osg::Matrix::translate(center[0] - 5.0, center[1] +
12.0, center[2] + 2.0);
			textTrans1->setMatrix(textpos1);
			textTrans1->addChild(textInLabGeode.get());
			root->addChild(textTrans1.get());
			std::cout << "this is in lab." << std::endl;
			text2 = true;
		}

		if (center[0] <= 4.0 && center[0] >= -4.0 && center[1] <= 10.0 && center[1] >=
0.0 && text2 == true){
			root->removeChild(textTrans1.get());
		}

		// User has touched the reactor, so activate the spray, smoke
		if (center[1] <= 4.0 && center[1] >= 2.0 && center[2] <= 4.0 && center[2] >= -3.0
&& once == false){
			root->addChild( updater.get() );
			root->addChild( sprayMt.get() );
			root->addChild(fire.get());
			once = true;
		}

		// if the User is not wearing mask, then add smoke will follow him
		if (once == true && gogglesOn == false){
			osg::ref_ptr<osg::MatrixTransform> smokeBurn = new
osg::MatrixTransform;
			osg::Matrix smokePos = osg::Matrix::translate(center[0], center[1],
center[2]);
			smokeBurn->setMatrix(smokePos);
			smokeBurn->addChild(fire.get());
			root->addChild(smokeBurn.get());
			osg::ref_ptr<osg::MatrixTransform> textTrans2 = new
osg::MatrixTransform;
```

```
						osg::Matrix textpos2 = osg::Matrix::translate(center[0] - 5.0, center[1] +
3.0, center[2] + 2.0);
						textTrans2->setMatrix(textpos2);
						textTrans2->addChild(textBlindGeode.get());
						if (text3 == false){
							root->addChild(textTrans2.get());
							text3 = true;
						}

				}
	}
	return 0;
}
```