A Project Report on

"3D SIMULATOR FOR DISASTER IMPACTS"

Submitted in partial fulfillment of the requirement for

Degree in Bachelor of Engineering (Computer Engineering)

By

Mr. Ajinkya Pisal

Mr. Javid Khan

Ms. Neeta Kokane

Guided by

Ms. Trupti Lotlikar

Department of Computer Engineering

Fr. Conceicao Rodrigues Institute of Technology

Sector 9A, Vashi, Navi Mumbai – 400703

University of Mumbai

2012-2013

CERTIFICATE

This is to certify that the project entitled

3D SIMULATOR FOR DISASTER IMPACTS

Submitted By

Mr. Ajinkya Pisal	100941
Mr. Javid Khan	100969
Ms. Neeta Kokane	100970

In partial fulfillment of degree of **B.E. (SEM VIII)** in **Computer Engineering** for term work of the project is approved.

External Examiner	**Internal Examiner**
___________________	___________________
External Guide	**Internal Guide**
___________________	___________________
Head of the Department	**Principal**
___________________	___________________

Date: - **College Seal**

ACKNOWLEDGEMENT

This project has been a result of the combined efforts of not only the members of the project group but also our teachers, family and friends.

Firstly, we would like to thank HOD, Computer department, Prof**. H. K. Kaura** and **Prof. M. Kiruthika**, for giving us opportunity and permission to undertake this project.

Our project guide ***Ms. Trupti Lotlikar*** provided us timely, excellent and continuous guidance throughout the semester. She supported and suggested new, constructive ideas and the necessary changes whenever required. Without her, the successful completion of the project would not have been possible.

Special thanks to our family and friends for being with us whenever needed and for their support. Without their co-ordination and support, the project would not have been possible.

ABSTRACT

Natural disasters like tsunami, earthquakes, and floods cause severe damage to human life and environment. These disasters and their impacts are extremely harmful to human life, hence is it necessary to analyse its effect and behaviour in order take preventive measures. But it is nearly impossible to simulate them in real world and calculate effects of disasters.

One solution to problem is to make use of shake table and wave basin to see effect of earthquake and flood respectively. But these systems are costlier and not available to wide range of people like city planner and city builders. To overcome this we are making use of computer graphics for simulation of earthquake and flood.

Our project is a 3D simulator for disaster impact. The user can design various 3D scenarios through an easy to use interface, which is concerned with type of terrain, building structure and their positioning. User can simulate two disasters namely flood and earthquake. The simulation is carried out by considering the building architecture and physical characteristics of the environment. After Observing and analysing the impact of the disaster user gets a clear idea of what actions to be taken in order to improve the construction approach or maybe change the location of construction. Hence our project can be used during the planning and designing phase of habitat construction to build a disaster proof city without spending a lot of money.

INDEX

LIST OF FIGURES

1. INTRODUCTION

In early days the cities were just a result of many buildings built within a certain radius but today's cities are well planned in terms of structure of roads, building heights, building positions etc. Also the soil testing and ground embankments are one of the major activities while deciding a building's size and location .One more important aspect that needs to be considered in construction planning is disaster proofing which can lead to a more safe and disaster proof habitat for human life but it is really challenging to predict the uncertainty of damage a disaster can cause.

Simulation of disasters is the best way to learn about the impacts and behavior of disasters. Simulation can be done by using fake building structures and applying forces based on the type of disaster. This process is very tedious and time consuming and such projects are handled by governments and are not accessible to general public. Our approach is to make use of 3D computer graphics and use virtual environment to simulate the impact of disasters.

1.1 3D Model

3D modeling is the procedure of developing a 3D model using specialized software. It is a process of creating a wireframe model that represents a three dimensional object. That object can be alive or inanimate. A three dimensional model is created using a set of points in 3D space, which are connected by various geometric data such as lines, and curved surfaces.

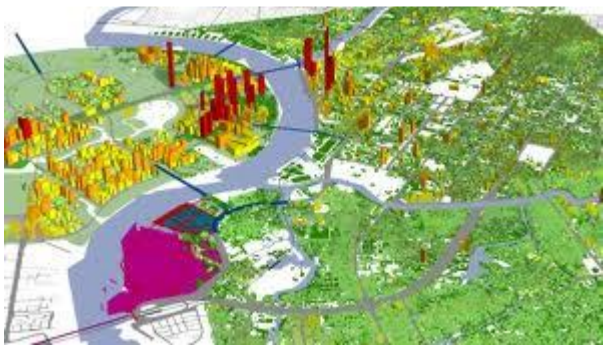


Fig 1: 3D City

Three-dimensional models are created using four methods:

- **Polygonal modeling**

  Many three dimensional models are created as textured polygonal models. Polygonal modeling is a method of creating a 3D model by connecting line segments through points in a 3D space. Polygonal models are very flexible and can be rendered by a computer very quickly.

- **Primitive modeling**

  This is the simplest way of modeling three-dimensional objects. Using geometric primitives such as cylinders, cones, cubes and balls, complex models are created. This approach ensures easy construction, as the forms are mathematically defined and precise. Primitive modeling is mainly used in developing 3D models of technical applications.

- **NURBS modeling**

  The NURBS (Non-uniform rational B-spline) modeling method can be found in popular software like Maya. The developer can create smooth-surfaced 3D models using this modeling technique. Unlike polygonal modeling techniques, which can only approximate curved surfaces using numerous polygons; NURBs modeling can truly create smooth curved surfaces.

- **Splines and patches modeling**

  These methods are similar to the NURBS modeling procedure. They depend on curved lines to identify the visible surface.

Advantages of 3D Modeling

- 3D modeling is beneficial for those instances in which we will benefit from seeing a physical object in a solid form. For this reason, 2D models cannot justify the results required from observing the model.

- A 3D model is a real, more realistic than 2D drawings.

1.2 Simulation

Simulation is the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model be developed; this model represents the key characteristics or behaviors of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over time.

- **Computer Simulation**

  A computer simulation or a computer model is a computer program that attempts to simulate an abstract model of a particular system.

- **3D computer graphics**

  3D computer graphics (in contrast to 2D computer graphics) are graphics that utilize a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering 2D images.

1.3 Disasters

A **disaster** is a natural hazard resulting in an event of substantial extent causing significant physical damage or destruction, loss of life, or drastic change to the environment.

1.3.1 Flood

Flood is overflow of water that drowns the land. It may due to accumulation of water on saturated area or because volume of water in river or lake overflows.

Types of Flooding

<u>Flooding can be divided into different categories according to their duration:</u>

1. Slow-Onset Floods

Slow-Onset Floods usually last for a relatively longer period; it may last for one or more weeks, or even months. As this kind of flood last for a long period, it can lead to lose of stock, damage to agricultural products, roads and rail links.

2. Rapid-Onset Floods

Rapid-Onset Floods last for a relatively shorter period; they usually last for one or two days only. Although this kind of flood lasts for a shorter period, it can cause more damages and pose a greater risk to life and property as people usually have less time to take preventative action during rapid-onset floods.

3. Flash Floods

Flash Floods may occur within minutes or a few hours after heavy rainfall, tropical storm, failure of dams or levees or releases of ice jams. And it causes the greatest damages to society.

<u>Flooding can also be divided into different categories according to their location:</u>

1. Coastal Floods

Coastal Floods usually occur along coastal areas. When there are hurricanes and tropical storms, which will produce heavy rains, or giant tidal waves created by volcanoes or earthquakes, ocean water may be driven onto the coastal areas and cause coastal floods.

2. Arroyos Floods

An arroyo is river that is normally dry. When there are storms approaching these areas, Fast-moving River will normally form along the gully and cause damages.

3. River Floods

This is the most common type of flooding. When the actual amount of river flow is larger than the amount that the channel can hold, river will overflow its banks and flood the areas alongside the river. And this may cause by reasons like snowmelt or heavy spring rain.

4. Urban Floods

In most of the urban area, roads are usually paved. With heavy rain, the large amount of rainwater cannot be absorbed into the ground and leads to urban floods.

Flooding can happen anywhere, but certain areas are especially prone to serious flooding.

The areas are as follows,

- **High Risk Areas**

  In high-risk areas, there is at least a 1 in 4 chance of flooding during a 30-year mortgage. All home and business owners in these areas with mortgages from federally regulated or insured lenders are required to buy flood insurance.

- **Moderate To Low Risk Areas**

  In moderate-to-low risk areas, the risk of being flooded is reduced but not completely removed. These areas submit over 20% of NFIP claims and receive one-third of disaster assistance for flooding. Flood insurance isn’t federally required in moderate-to-low areas, but it is recommended for all property owners and renters.

- **Undetermined Risk Areas**

  No flood-hazard analysis has been conducted in these areas, but a flood risk still exists. Flood insurance rates reflect the uncertainty of the flood risk.

1.3.2 Earthquake

Earthquakes are natural phenomena caused by the sudden release of tension from the rocks in the Earth’s Crust and upper mantle, resulting in potentially devastating seismic waves.

Earthquake Hazards

Following are the earthquake hazards,

- **The Effect of Ground Shaking**

  The first main earthquake hazard (danger) is the effect of ground shaking. Buildings can be damaged by the shaking itself or by the ground beneath them settling to a different level than it was before the earthquake (subsidence).

- **Ground Displacement**

  The second main earthquake hazard is ground displacement (ground movement) along a fault. If a structure (a building, road, etc.) is built across a fault, the ground displacement during an earthquake could seriously damage or rip apart that structure.

- **Flooding**

  The third main hazard is flooding. An earthquake can rupture (break) dams or levees along a river. The water from the river or the reservoir would then flood the area, damaging buildings and maybe sweeping away or drowning people.

- **Fire**

  The fourth main earthquake hazard is fire. These fires can be started by broken gas lines and power lines, or tipped over wood or coal stoves. They can be a serious problem, especially if the water lines that feed the fire hydrants are broken, too.

1.4 Disaster Simulator

A **Disaster simulator** is software that virtually re-creates real world cities and various aspects of the natural environment. Depending on their purpose, disaster simulations employ various types of software, modeling detail and realism. They can range from 2D models to realistic 3D models.

This includes the equations that govern how earthquake and flood takes place, how they react to disasters, and how they react to external environmental factors such as roads, buildings, human beings, natural environment, etc. Disaster simulation is used for a variety of reasons, including educational learning, the design and development of the city, and research into disaster characteristics and control handling disasters.

2. LITERATURE SURVEY

Over the last few decades, the frequency and intensity of natural disasters have risen significantly. In addition to a more volatile natural environment, aging urban infrastructure and an increasingly complex and interdependent network of technological systems have created a multitude of hazards to which humans are vulnerable. Within this more hazardous environment, the impacts of disasters have also risen sharply, disrupting the lives of those affected and causing unprecedented property damage and loss.

The system for disaster simulation has been improved dramatically over the decade. First eras of disaster simulator were the involving real world scenarios on which simulation of natural disasters on small scale were done. Then with the increase in computing capabilities of the computers came the simulators. First 2D simulators were developed. 2D simulators were not giving accurate representation of the real life scenario. Then came the 3D simulator with more realistic view of the environment. But 3D simulators became more powerful and realistic with the introduction of the GIS. GIS provided the necessary geographical data, which is essential part for the simulation of the disasters.

1) Real World Disaster Simulators

The earlier systems for implementing disaster and impact analysis were simulating natural disaster at small scale in real world.

A. Shake-Table Testing [2006]

Earthquake simulation was known as shake-table testing [1]. Shake table is a device for shaking structural models or building components with a wide range of simulated ground motions, including reproductions of recorded earthquakes time-histories. While modern tables typically consist of a rectangular platform that is driven in up to six degrees of freedom (DOF) by servo-hydraulic or other types of actuators, the earliest shake table, invented at the University of Tokyo in 1893 to categorize types of building construction, ran on a simple wheel mechanism. Test specimens are fixed to the platform and shaken, often to the point of failure. Using video records and data from transducers, it is possible to interpret the dynamic behavior of the specimen.

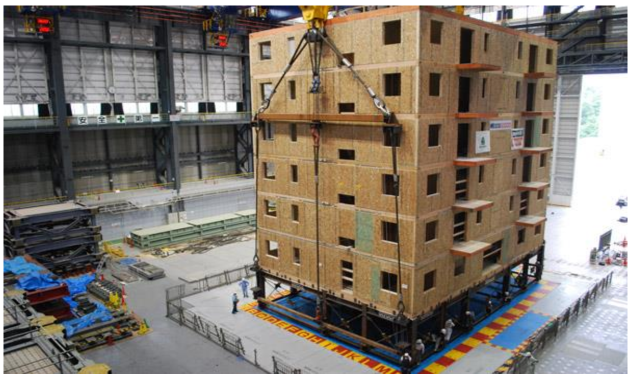


Fig 2: shake-table test model

The simulation test method consists of the physical and numerical parts actively interacting with each other during the test. The first part consists of an experimental test specimen that undergoes dynamic excitation from actuators as if under the force of an earthquake. The second part is a computational subsystem, which models the remainder of the test structure.

Disadvantages

The drawback of using shake tables is that they are very expensive to build and payload is limited to the size of the table and its payload capacity. Expenses increased greatly when building larger shake tables with multi-degree of freedom capabilities, so the reduction in scale required for models remains a major drawback for using shake tables for seismic testing.

B. Tsunami wave basin [2003]

The Tsunami Wave Basin is designed by O.H. Hinsdale Wave Research Laboratory as a shared-use facility to understand the fundamental nature of tsunami inundation and to improve the numerical tools for tsunami mitigation:

• Tsunami inundation and overland flow

• Tsunami-structure impact

• Tsunami debris flow and scour

• Harbor resonance

In addition to tsunami research, the facility is used for general testing of coastal infrastructure and for near shore processes research.

Fig 3: Tsunami wave basin

Disadvantages

The problem is the size of the basin as tsunami waves can reach a height of more than 20 foots .It also requires a lot of water and hydraulic power while generating the waves. The required manpower is high and the reading are to be taken manually which might night be accurate.

2) 2D Flood Simulator [2006]

FLO-2Ddeveloped by is a mathematical model for numerical simulation of 2D flood waves due to complete or partial dam-break [2]. The governing water equations are solved by an implicit diagonal numerical scheme, based on the Mac Cormack's predictor-corrector technique. The mathematical model is used to numerically compute the water surface from supercritical flow to sub critical one or from sub critical flow to supercritical one in a rectangular open channel, and the numerical results are compared with the theoretical results. And it is also used to predict 2D flood waves due to partial instantaneous dam-break in a rectangular open channel with three rectangular cylinder barriers downstream, and the reliability of numerical results is analyzed. The comparison and the analysis show that the proposed method is accurate, reliable and effective in simulation of dam-break flood waves.

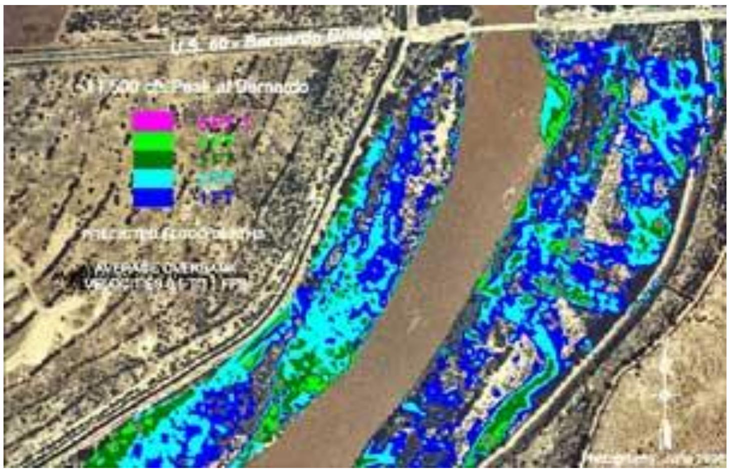


Fig 4: Flo-2D Flood Simulator

Disadvantages

The simulator was quite efficient but the data processing required too many mathematical calculations. Also the 2D visualizations were not so impressive as it could not portray the depth of the scene.

3) 3D Disaster Simulators [2009]

The 3D disaster simulator relies on a physics engine and built in artificial intelligence to provide realistic, 3D emergency situations. The Disaster scenarios include algorithms, which take into account: type of threat, terrain, buildings, trees, and rivers.

A. Earthquake Simulator

Hercules is a 3D earthquake simulator developed by Ricardo Taborda and Jacobo Bielak from Carnegie Mellon University [3]. Hercules combines an octree-based approach for generating large finite- element unstructured meshes (with hundreds of millions to billions of elements) and implements highly efficient and scalable algorithms to solve the wave propagation problem embedded in its formulation for earthquake simulations caused by kinematic faulting. Hercules also includes recently developed features to model more realistic complex engineering systems of interest to earth- quake engineers, such as material plasticity and the incorporation of multiple simplified building models.

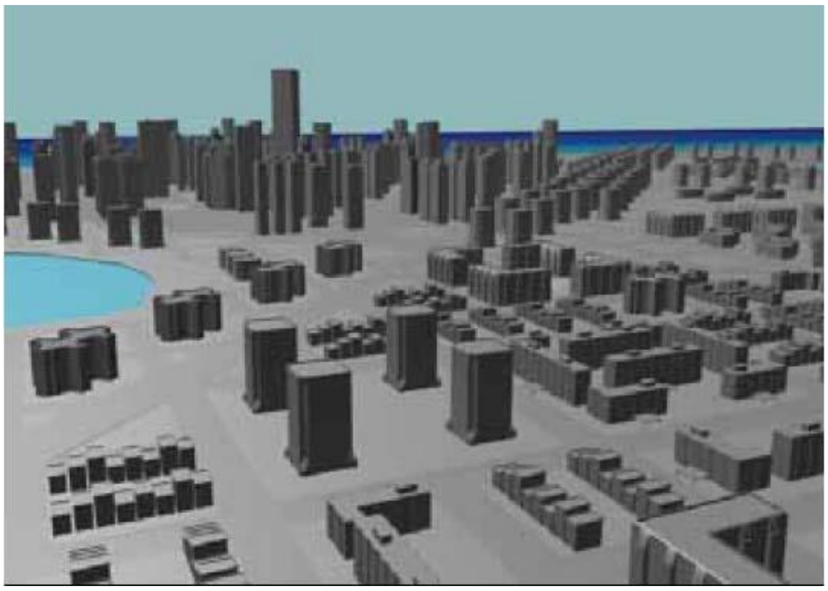


Fig 5: Hercules-earthquake simulator

Advantages

It allow us to,

- •Study design and architecture of urban areas,
- •Understand soil-structure and site-city interaction effects on the ground motion due to the presence of the built environment, and
- •Obtain better estimates about buildings' behavior during earthquakes.

B. Flood Simulator

Australian and global media used AAM's 3D flood simulation of Brisbane extensively during the recent flood crisis in Brisbane [5]. The flood level simulations were produced interactively in 3D. This was a very effective way of communicating the impact of the flood on buildings and infrastructure and it provided a valuable tool in assessing the potential risk for the city in the days leading up to and during the flood peak. The flood simulations were featured extensively on Australian and international media. The videos supplied residents, business owners and the public with realistic representations of the predicted flood event. The flood simulation was based on accurate geospatial terrain data created by AAM. The company provides engineers and planners with photo realistic 3D city models and highly accurate terrain models to aid in their assessment and design tasks. AAM General Manager, Brian Nicholls said he hoped that AAM's simulations helped in some way to communicate the scale and extent of the flooding.

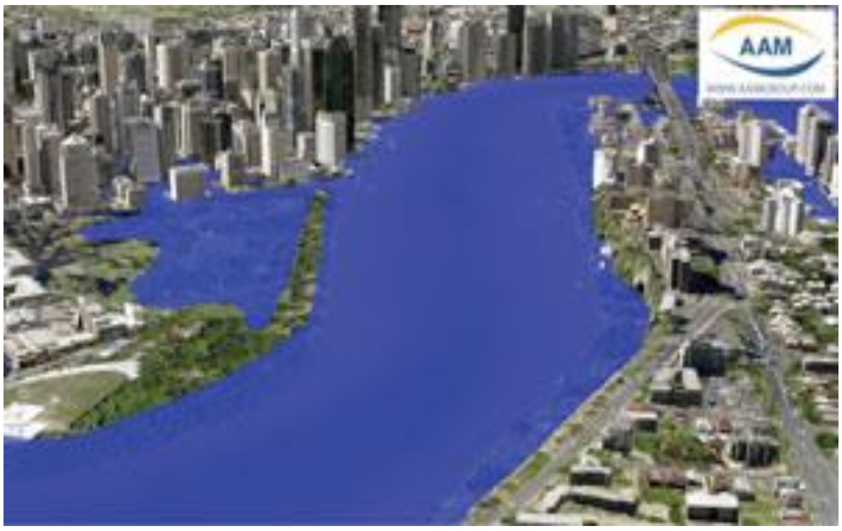


Fig 6: AAM’s 3D flood simulation

4) 3D Disaster Simulator with GIS [2011]

The recent trend in the disaster simulator is the use of GIS, which allows for building of realistic simulation models. This system makes use of the multi-agent model considering geographical information. Example of such system is “artisoc” developed by Keisuke Uno, Kazuo Kashiyama at Chuo University, Japan [6].

ArtiSoc consist of the mainly three parts. First part consists of the modeling of the land and buildings by using geographical information system (GIS). Second parts consist of the analysis of disaster using multi-agent model and third part consist of the visualization of the numerical results using virtual reality and suggestion for disaster evacuation.

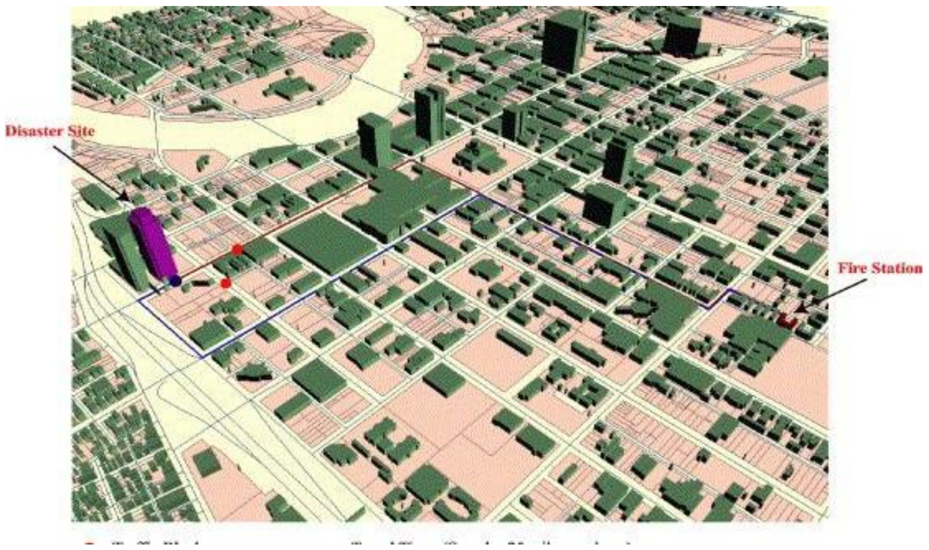


Fig 7: Artisoc-disaster simulator using GIS

The System makes use of ArcGIS software for GIS, POV-ray for 3d animation and Visual Basic for building GUI application. The software allows user to change parameter according to his need form panel on the screen. The main parameters are walking distance, velocity of refugee and evacuation time after disaster.

Then based on the refugee's position is makes road network. Then based on this data is makes use of the Dijkstra's shortest path algorithm to find the minimum distance for evacuation of humans.

Advantages

Easy to change scenario with GUI and change various parameters to see their effects. It is possible to evaluate not only the damage of structure but also the damage of human being. By using the visualization based on virtual technique, the high quality CG image has been created. From this, the user can understand the feeling of refugee.

Disadvantages

The scenarios look less realistic as the modeling capability is limited by the GIS system. Also the effects can only be shown by imposing some graphics on the environment.

3. EXISTING SYSTEM

UESS (Urban Earthquake Simulation System)

UESS is earthquake simulation system for urban areas. Urban earthquake simulation system (UESS) uses GIS as the model source, CAD as the model generating tools, FEA as damage prediction, and VR as the simulation platform. UESS generated the 3D structural model automatically based on GIS data and passes them to FEM for calculating of damage done on building due to specified earthquake. Finally performance of each building structure against specified earthquake is shown as output of the system.

UESS transforms 2D GIS data into 3D models by making use of automatic model generation function in UESS. It does it so by first drawing 2D outline from GIS database then according to height of the building in database, it constructs 3D building models. Then to make buildings much more realistic, Auto-texture module was developed. It applies the texture to building from more than 100 texture templates.

It visualizes damage on building by following procedure.

1. Identifies type of structure use for building
2. Estimate vibration period for building based on earthquake intensity
3. Based on mass and elastic stiffness of building, calculate center of gravity for each building
4. Based on shear force applied on building, destruction of building is displayed

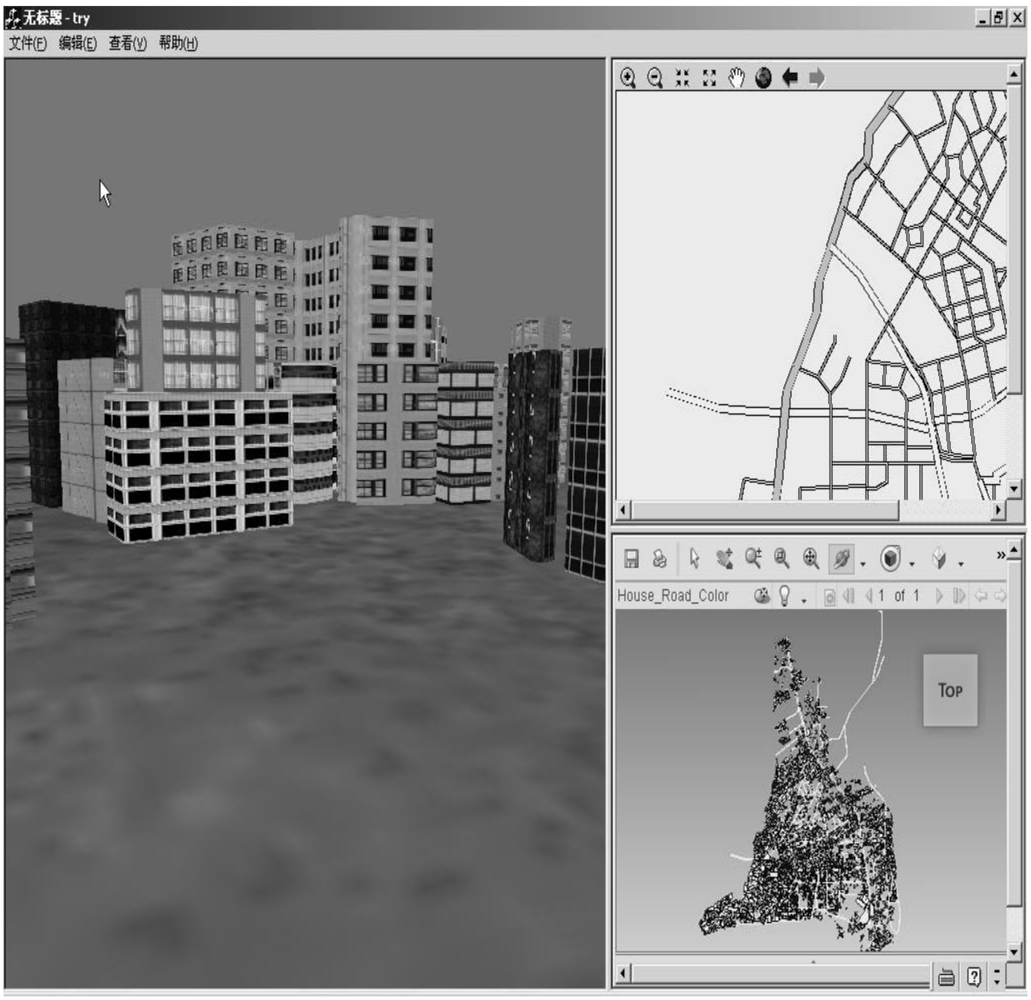


Fig 8: UESS Interface

Advantages:

1. More accurate real world data because of use of GIS database
2. Making use of GIS allows for simulating earthquake on real world places
3. 3D models give realistic appeal to simulation
4. FEM allows details visualization of where building bends and breaks

Disadvantages:

1. Focuses on earthquake simulation only (no simulation of flood)
2. Only shows impacts of earthquake on building, no suggestions are given to improve design of buildings
3. No rescue plan for safe evacuation of humans

4. PROBLEM STATEMENT

Natural disasters such as earthquake and floods cause severe damage to human lives and the environment, resulting in the need for mitigation. The method of mitigation should be efficient and cost effective.

The existing tools for estimating the impact of these disasters on cities are Shake-Table and Wave Basin. However, building these systems is costly and even then, the benefit is limited. Moreover, analysis of the tools and the interpretation of results require sound domain knowledge.

Therefore, cheaper solutions, which can be widely adopted, are the need of the hour to minimize the effects of disasters on cities. We propose a computer simulation technique to calculate and estimate the disasters impacts, one, which would surpass the accuracy of aforementioned tools and yet be cost-effective.

5. PROPOSED SYSTEM

Proposed system is a 3D simulator for Disaster Impacts. It will be used for simulating disaster impacts on scenarios created by user and will also provide suggestions about the city's construction planning to make it more disaster proof.

Various Existing systems observed during the literature survey mainly calculate the disaster impact by using environmental and building parameters. And effects of disaster are shown using 2D and 3D graphic rendering. Our proposed system will present the simulation in 3D& more realistic way. The existing systems basically focus on a single disaster simulation on one environment, while the proposed system will simulate two disasters on a single environment.

The environment designing and editing will be flexible to a great extent, allowing the user to create more realistic and customizable environment for simulation. In proposed system user will be able to create his own scenarios. He can choose to create building on any location in terrain. While creating building he has to specify parameters like building name, location of building in 3D space, type of foundation used by that building.

For simulating earthquake, system take into consideration parameters like the type of foundation used by building, type of soil, distance of building from the epicenter and earthquake intensity. Then based on these parameters system calculates the peak ground acceleration and apply seismic forces on building according to its mass.

For simulating flood, system takes into consideration parameters like the type of soil in area, run off coefficient of soil, duration of rain and rain intensity. Then based on these parameters system calculates the increase in water height in area.

After the simulation a list of suggestions will also be provided that can be enforced on the city construction to make it disaster proof.

6. SCOPE

The system will consist of 3D models which represent a 3D object using a collection of points in 3D space, connected by various geometric entities such as triangles, lines, curved surfaces, etc. In 3D computer graphics, 3Dmodeling is the process of developing a mathematical representation of any three-dimensional surface of object via specialized software.
3D models will be created for two disasters such as Earthquake and Flood. The system will provide a user a visual editor, in which user will get following two options,

1. Load default or saved Scenario
2. Create new scenario

In create new scenario user can create building by specifying the number of floors, foundation type and its position in 3D terrain. Then system will simulate the scenario that he/she has made, according to the option selected by the user. System will display the required suggestions about the selected disaster. System also keeps a log report of all the consequent simulation done by the user so that the user can compare the results of previous runs with the current run. The proposed system will help a user to understand the impacts and suggestions about the particular disasters (Earthquake, flood) in 3D model in a realistic way. So such system can be used for city planning and construction.

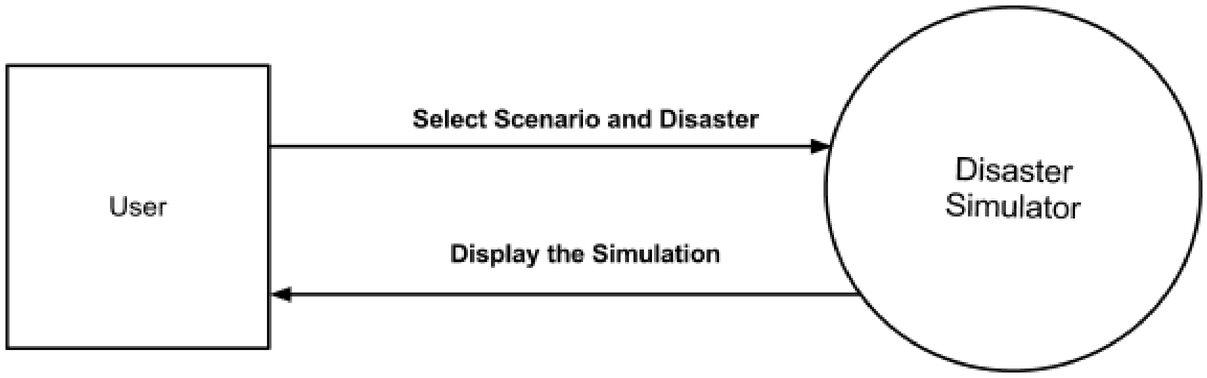
Advantages:

- A 3D model is a real, more realistic than 2D drawings
- Easy to use interface for creating, deleting and repositioning a building
- Obtain better estimates about impacts during Earthquake or Flood
- Keeps a log report of all the simulations done by the user
- Suggestions about the disaster (Earthquake, Flood)

7. DESIGN

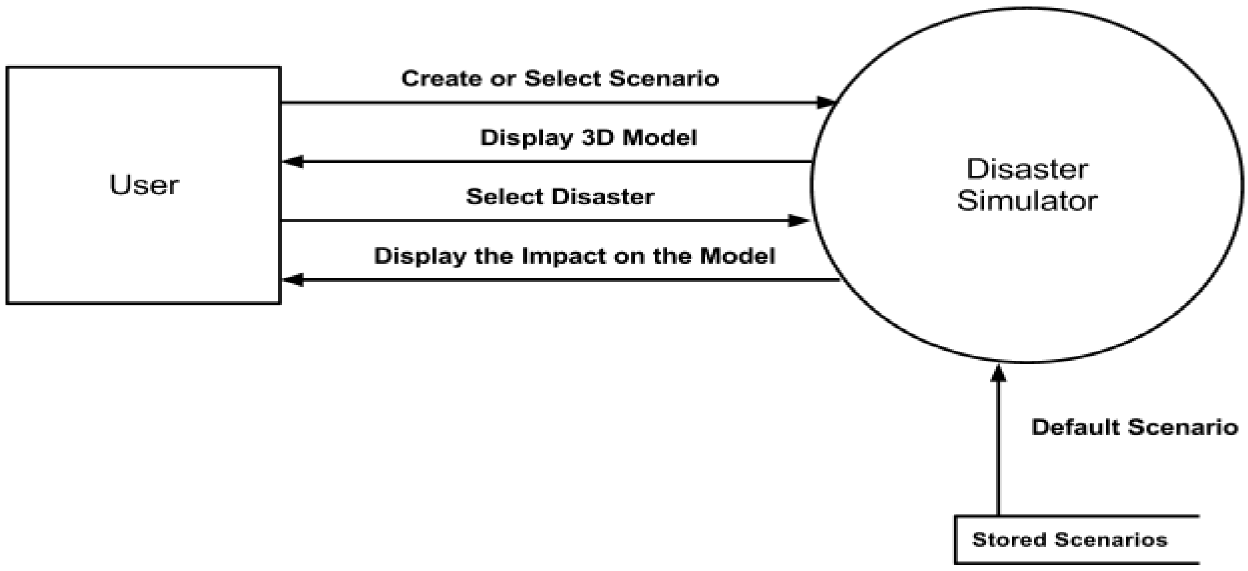
7.1 Data Flow Diagram:

<u>**Context Level DFD:**</u>



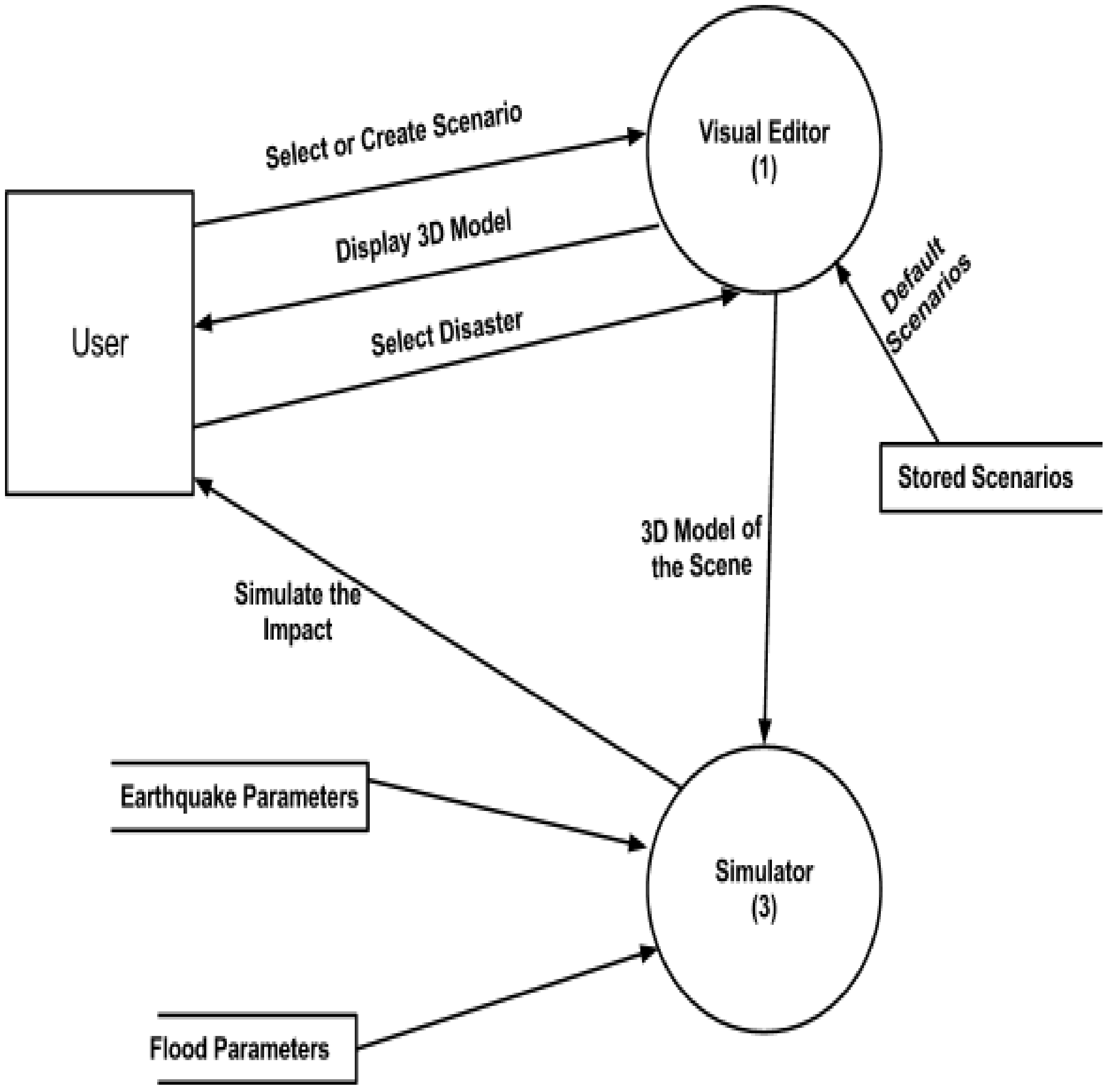
The context level DFD represents the entire system as a single bubble. The system takes the scenario and disaster type as input and displays the simulation result. The scenario can be either user created or a default scenario according to user choice.

<u>**Level-0 DFD**</u>



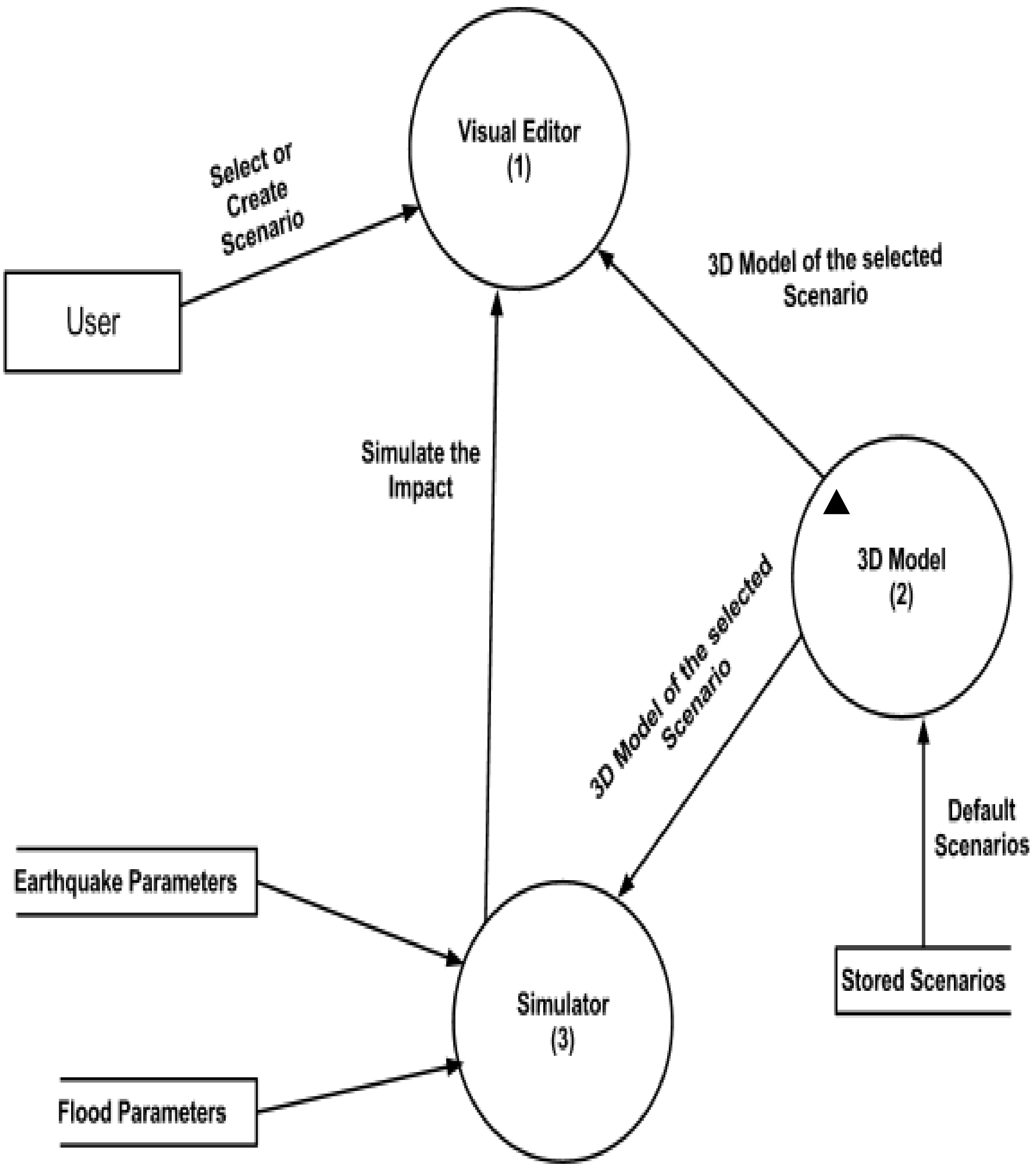
Level 0 DFD shows the storage specification of import and export utility and refines the user actions into more specific steps as shown above. The data store is nothing but basic disk storage than can store an xml file. No specialized database is required.

Level-1 DFD:



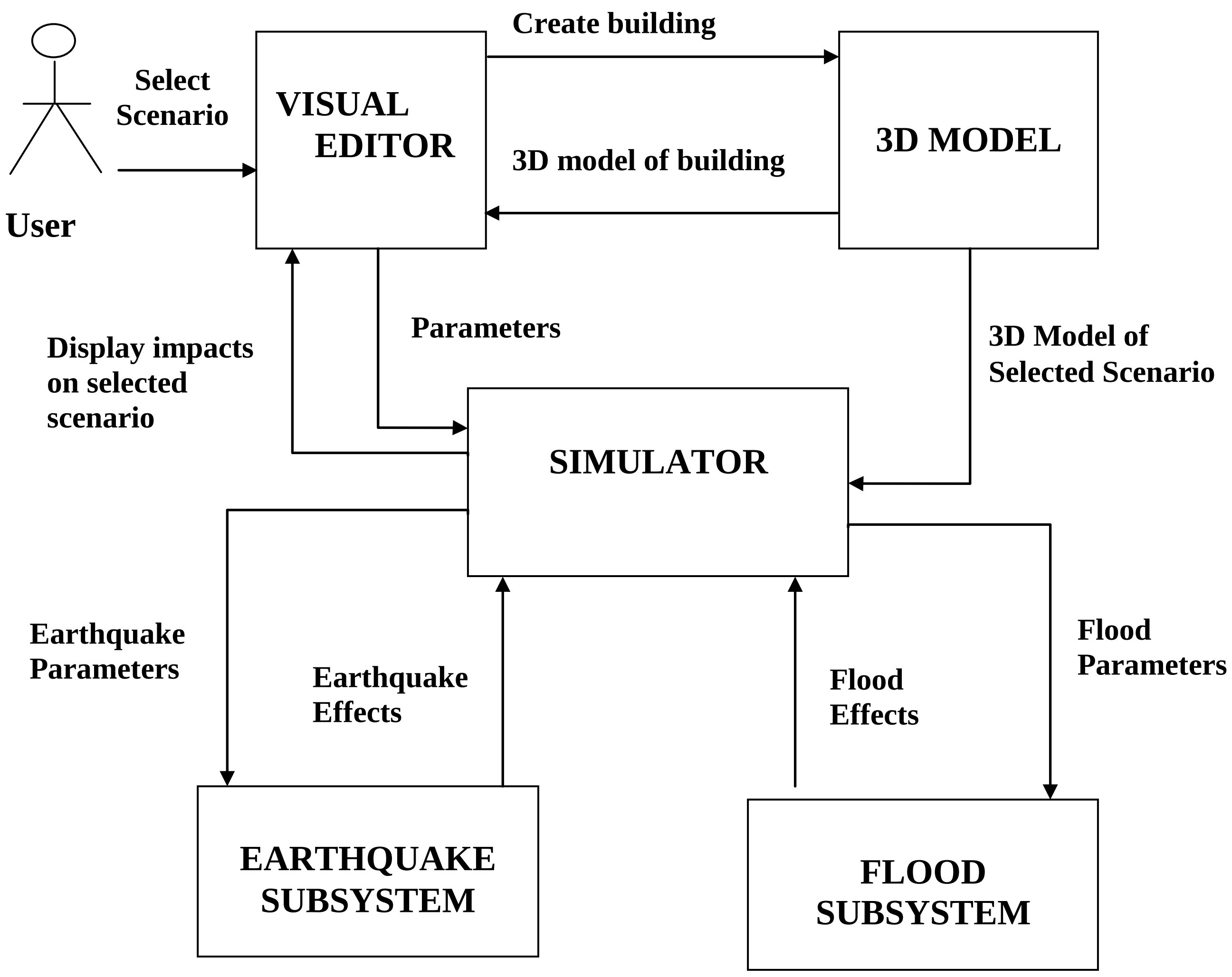
Level-1 DFD divides the simulation system into 2 modules namely Visual Editor and Simulator. The visual editor deals with user interactions and allows the user navigate and to create 3D scenarios. The simulator takes in earthquake and flood parameters and performs calculations based on the some formulas. These formulas give the amount of force or the rise in water level for earthquake and flood simulation respectively.

Level-2 DFD:



The level-2 DFD is the most refined DFD which represents the entire system as 3 different modules namely Visual Editor, Simulator, 3D Model .The 3D model generation system is separated from the simulator as an independent module that deals with dynamic mesh creation and rendering. The import of stored scenarios is also attached to the 3D model hence allowing the simulator module to focus on the simulation.

7.2 Block Diagram:



The block diagram shows all the modules present in the system and the flow of information and object between them. The various modules servers for following purposes:

1) Module 1- Visual Editor

It provides GUI, wherein user can construct scenario. Constructing scenario in Visual Editor is a placing of building on terrain at location specified by user. With help of editor user also provides parameters for building, flood and earthquake.

2) Module 2- 3D Model

This module is used for creation of 3d scenario on which user will be simulating specified disaster. It constructs 3D model of selected components from visual editor by calling respective 3D model generation function.

3) Module 3- Simulator

This module forms the core mechanism of the disaster simulator. It calculates the disaster impacts on user-defined scenario. It takes 3D Model of selected scenario from 3D Model module, and also it takes the earthquake or flood parameters of disaster as given by user to simulate earthquake or flood. Then these parameters are provided to respective disaster system. It simulates the scenario and provides impacts and suggestions on visual editor. It consist of two main subsystems as given below,

- **Earthquake Subsystem**

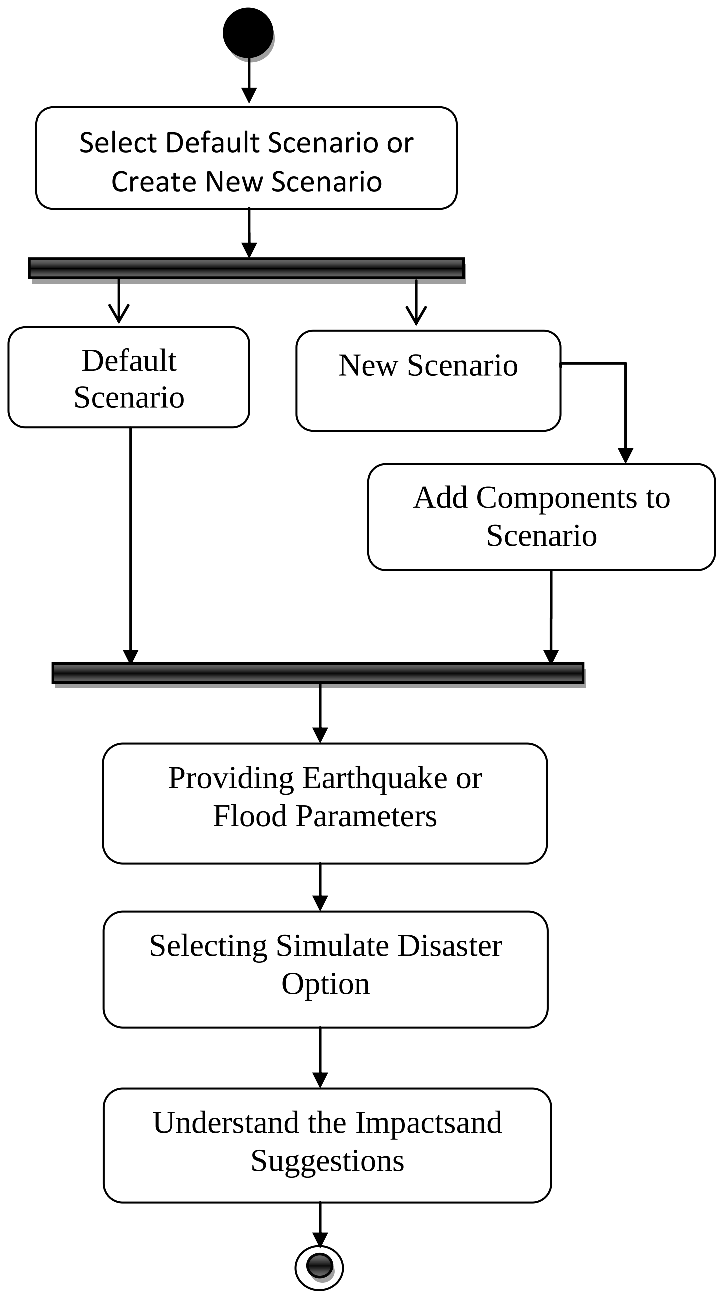
It takes in Earthquake parameters from user with the help of visual editor. Then based on the parameters provided, this system calculates impact on the user-defined scenario.

- **Flood Subsystem**

It takes in Flood parameters from user with the help of visual editor. Then based on the parameters provided, this system calculates impact on the user-defined scenario.

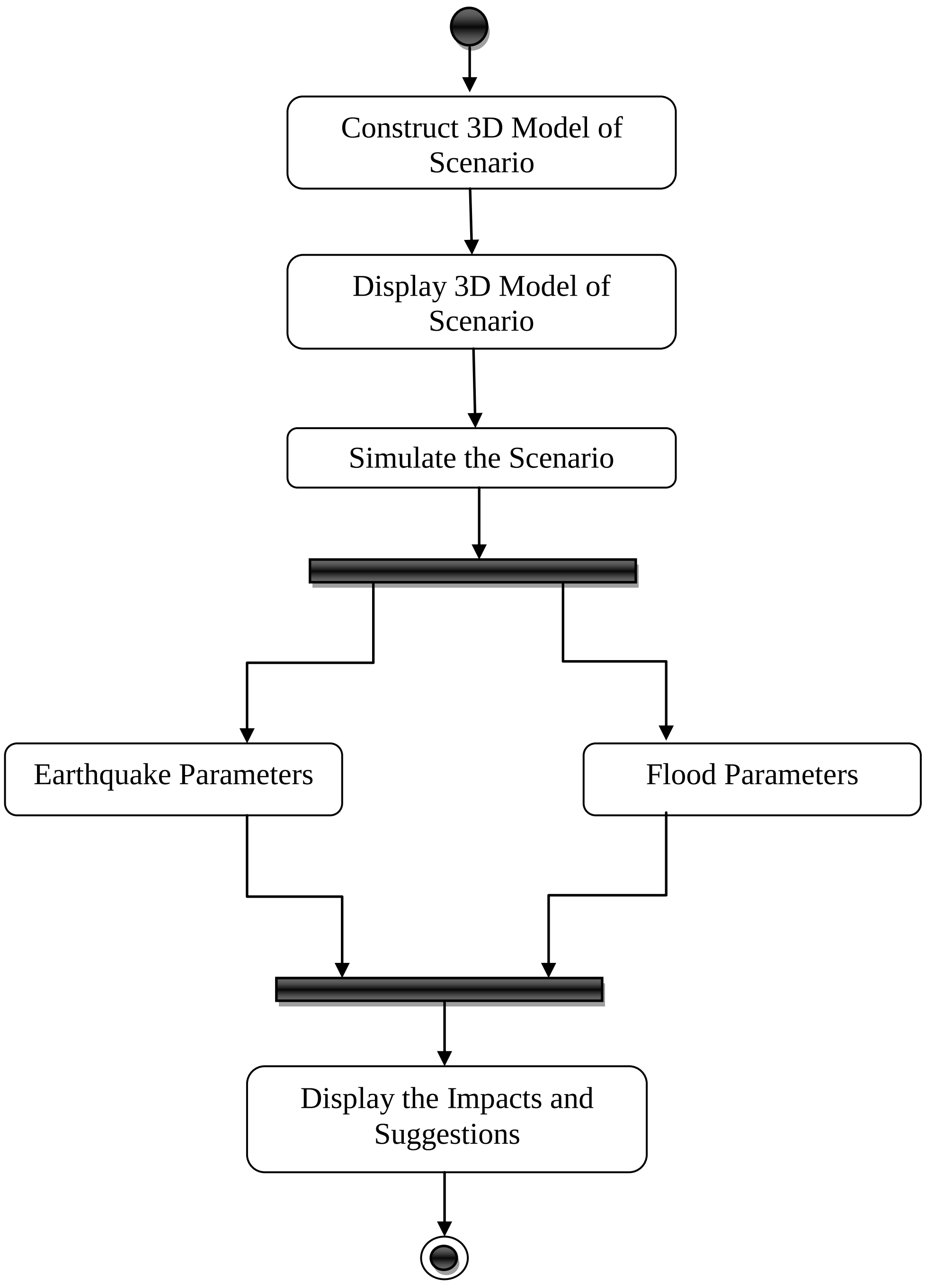
7.3 Flow Chart:

User Activity:


Select Default Scenario or Create New Scenario
Default Scenario
New Scenario
Add Components to Scenario
Providing Earthquake or Flood Parameters
Selecting Simulate Disaster Option
Understand the Impactsand Suggestions

User is first prompted to select a scenario. User either selects one of the already saved default scenarios or creates new scenario. New scenario can be created by simply specifying the name of building, number of floors, position and foundation type. Once the scenario is complete, user selects on of the disasters i.e. earthquake or flood. After that user hits simulate button and visualizes the impact of the selected disaster and reviews suggestions.

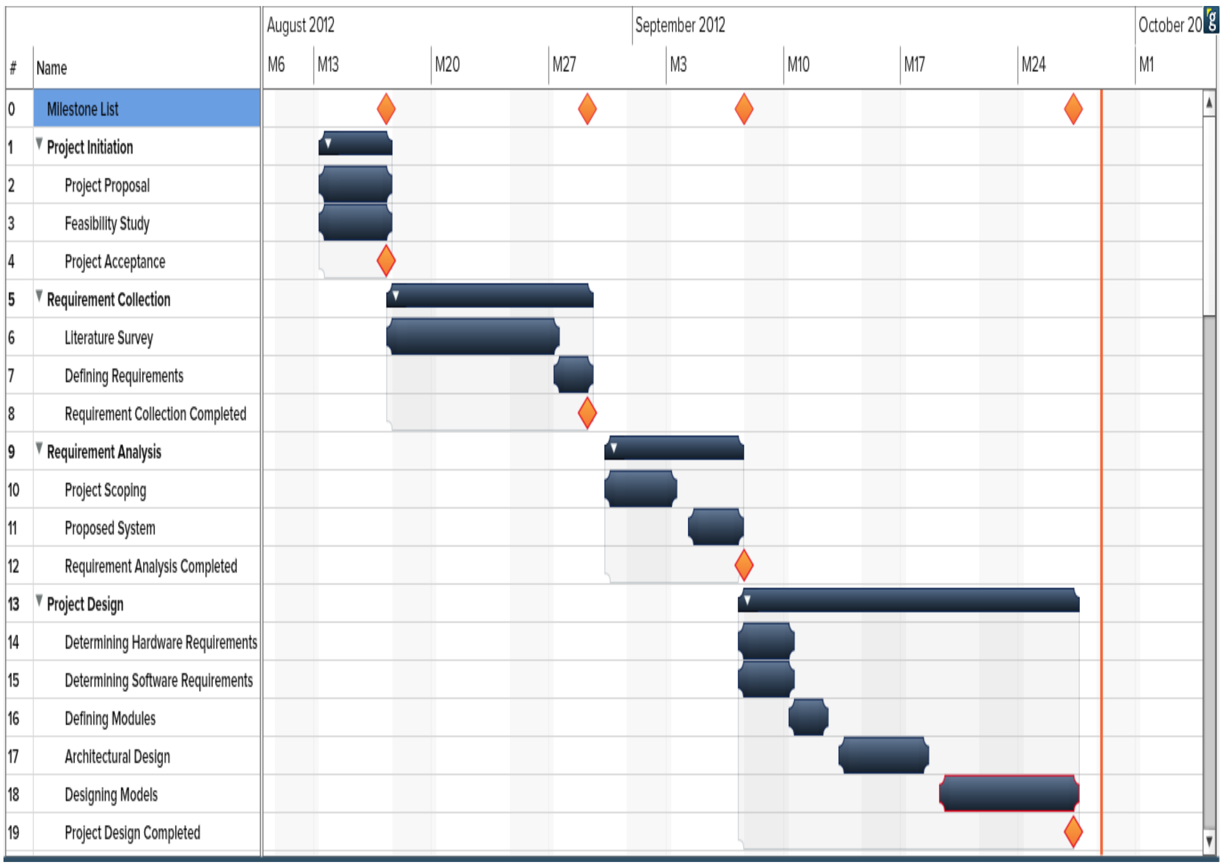
System activity:


Construct 3D Model of Scenario
Display 3D Model of Scenario
Simulate the Scenario
Earthquake Parameters
Flood Parameters
Display the Impacts and Suggestions

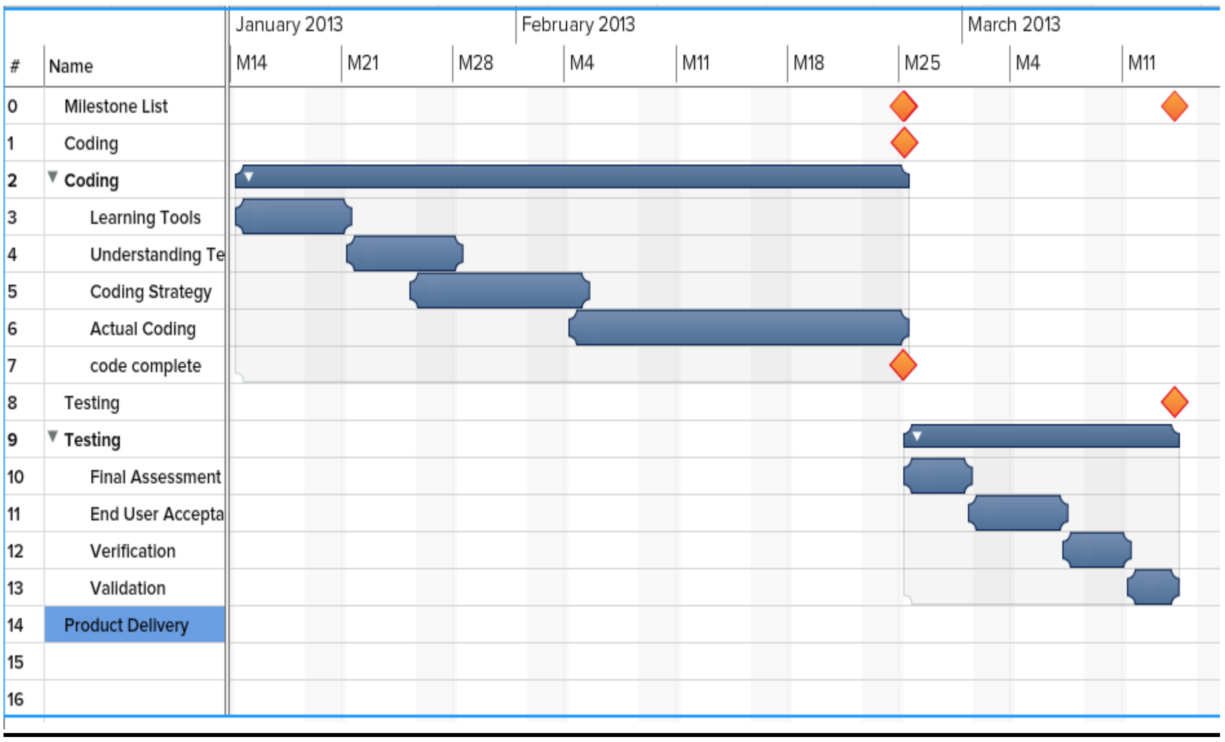
According to the scenario constructed by the user, system will construct the 3D model of it and display it on visual editor. Then based on the selected disaster and various parameters associated with that disaster, system will calculate impacts on scenario and will show suggestion in dialog box of visual editor.

7.4 Gantt Chart:

Project Planning:



Project Execution:



8. HARDWARE AND SOFTWARE REQUIREMENTS

Hardware Requirements:

Processor	**:**	**1 GHz**
Memory	**:**	**512 MB**
Video Card	**:**	**NVidia: Geforce2 or higher required**
		ATI: Radeon 7500 or higher required

Software Requirements:

OS	**:**	**Windows 7**
IDE	**:**	**Visual Studio 2010**
Language	**:**	**C++**
Rendering Engine	**:**	**OGRE 3D**

MFC:

The Microsoft Foundation Class Library (also Microsoft Foundation Classes or MFC) is a library that wraps portions of the Windows API in C++ classes, including functionality that enables them to use a default application framework.

OGRE 3D:

OGRE (Object-Oriented Graphics Rendering Engine) is a scene-oriented, real-time, flexible 3D rendering engine written in C++ designed to make it easier and intuitive for developers to produce applications utilizing hardware-accelerated 3D graphics.

SHADERS:

Shader is a computer program that runs on the graphics processing unit and is used to do shading the production of appropriate levels of light and darkness within an image - or, in the modern era, also to produce special effects or do post processing.

BULLET PHYSICS:

Bullet is an open source physics engine featuring 3D collision detection, soft body dynamics, and rigid body dynamics. It is used in games, and in visual effects in movies.

9. IMPLEMENTATION

9.1 Implementation design

For the implementation of the system we have used Ogre 3D for graphics rendering and Bullet Physics for earthquake simulation and Shaders for water simulation. Application is coded in C++. It has been developed using Microsoft Visual Studio 2010. For developing GUI of application we have made use of MFC. MFC is an application framework for creating windows applications.

To start with implementation first step was to make connection between MFC and Ogre 3D.It is done with passing handle of Ogre 3D to MFC window. After that we setup Ogre 3D in application with default configuration. With this a basic application with a black window is created.

Next step was to create whole application. For that purpose we made use of incremental approach. In increments we first created terrain and then added functionality of navigating in 3D space. After that we added functionality for creating building and managing building. After this 3D model and visual editor part was complete.

For implementing core part of the system we needed Bullet physics and Shaders. So we first made connection between Ogre 3D and Bullet Physics library. Then for simulating earthquake, we converted 3D model of building into rigid body using Bullet Physics. Then we implemented seismic forces for simulating earthquake. After that we created water, rain and completed flood simulation part of the system. At the end by making use of Bullet Physics library for earthquake simulation and Shaders for flood simulation we completed simulation module.

Next step was to provide functionality for saving and loading scenario. For strong scenario we have use xml with tags defined by us. Similarly we added functionality for loading scene from user created xml file.

After whole system was completed, we added validation for checking wrong inputs. Also during development we had tested application and using integration testing we had added functionality to system step by step. All of this incremental progress and testing resulted into a complete simulator for earthquake and flood simulation.

9.2 Implementation issues

During the implementation of our system we came across the following issues:

1. GUI

For creating GUI for application we have consider using wxWidget and QT. But they were not supported for newer version of Ogre. So we decided to make use MFC.

2. Water simulation

For water simulation we had first though about making use of NVidia’s PhysX library. But problem was that it runs only on computers having NVidia Graphics Card installed on them. So to overcome this problem and to get fluid simulation running on both AMD and NVidia Graphics card we made use of Shaders for simulating of water so that water simulation could run on any computer with graphics card supporting Shaders.

3. Physics integration

For simulation of earthquake it was necessary to convert 3d models of building to physical model. To do this we had consider using Bullet Physics Library but as it being a discrete library we could not use it directly with Ogre 3D. Solution was to use OgreBullet. OgreBullet is a wrapper, which does the connection between Ogre 3D and bullet. But again problem here was that OgreBullet does not provide all the functionality of Bullet Physics. So to overcome this entire problem we made the use of BtOgre. BtOgre is also a wrapper but it converts Ogre Scenenodes to rigid body and vice versa. Hence you get to use full functionality of the Bullet Physics and because of this Earthquake simulation was possible.

4. Terrain integration in MFC

For initialization of the terrain and terrain configuration object requires the use of *OGRE_NEW* instead of *new*. But use of *OGRE_NEW* was giving error because in MFC debug mode it defines *new* with *DEBUG_NEW* keyword. So it was overriding declaration of *OGRE_NEW* with *DEBUG_NEW*. Hence solution to this problem was deleting declaration of *DEBUG_NEW*.

9.3 Screenshots

We have developed our application in MFC using Visual Studio 2010 to implement our system. The screenshots of system are given below:

9.3.1 Scene Menu

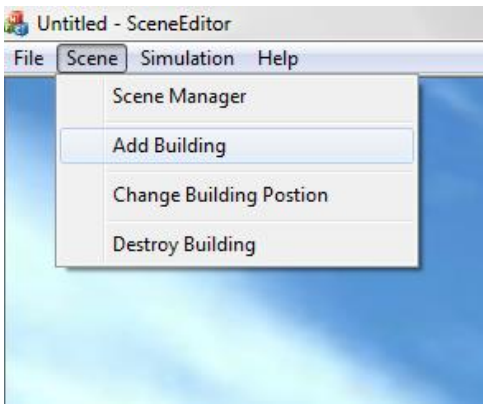


Fig 9: Scene Menu

This is the Scene menu of Disaster Simulator application. It has four menu options. Each of those option helps in managing buildings in 3D scenario.

9.3.2 Scene Manager

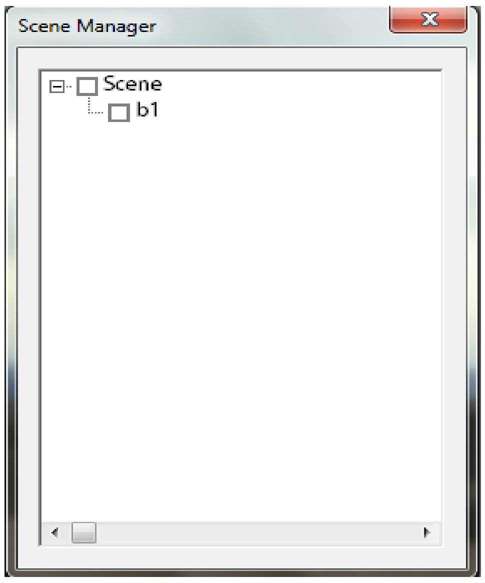
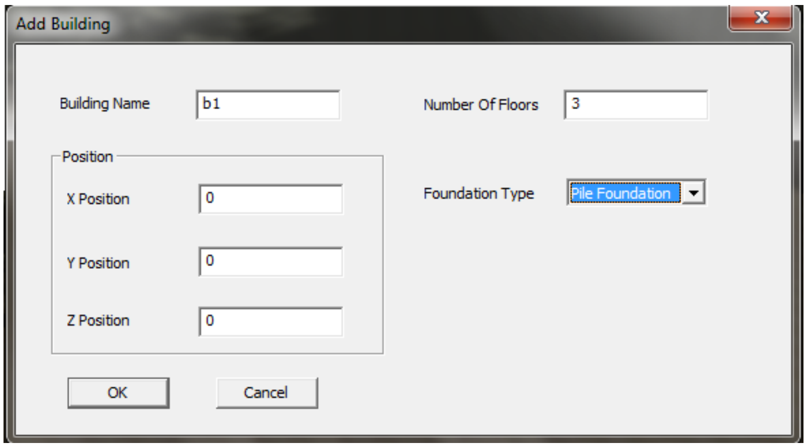


Fig 10: Scene Manager Dialog Box

It keeps track of all building present in scenario and it also helps in selecting particular building in scenario so that operations like changing building position, deleting building can be performed.

9.3.3 Adding Building

Clicking on Add building opens a new dialog box. As shown in the figure below,


Add Building

Building Name: b1

Number Of Floors: 3

Position

X Position: 0

Y Position: 0

Z Position: 0

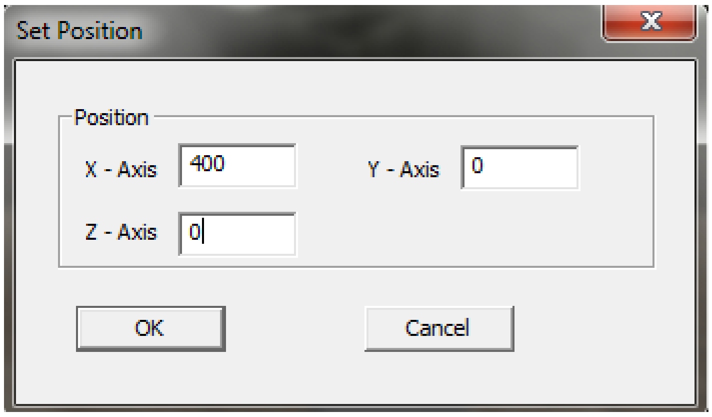
Foundation Type: Pile Foundation

Fig 11: Add Building Dialog Box

This dialog box is used for creating 3D model of the building. It takes several parameters from user and based on it create 3D model of building on terrain. First parameter is Building Name is used to identifying building in scenario. Next parameter is number of floors; it specifies the number of floors of building user wants to build. Next parameter is foundation type use for building. Here user can select any one foundation out of 3 foundation viz. Pile Foundation, Shallow Foundation and Raft Foundation. Next parameter is the position of the building in 3D space.

9.3.4 Changing Building Position

Clicking on Change Building Position opens a new dialog box. As shown in figure below,



X - Axis: 400
Y - Axis: 0
Z - Axis: 0

Fig 12: Change Building Position Dialog Box

It takes the new position of the selected building from Scene Manager. And translates building to new position as inputted by user.

9.3.5 Deleting Building

Clicking on Destroy Building deletes the selected building from Scene Manager.

9.3.6 Simulation Flood

For simulating flood simulation, user has to select Run option from Simulation Menu. Then click on Flood simulation and then on Start Flood.

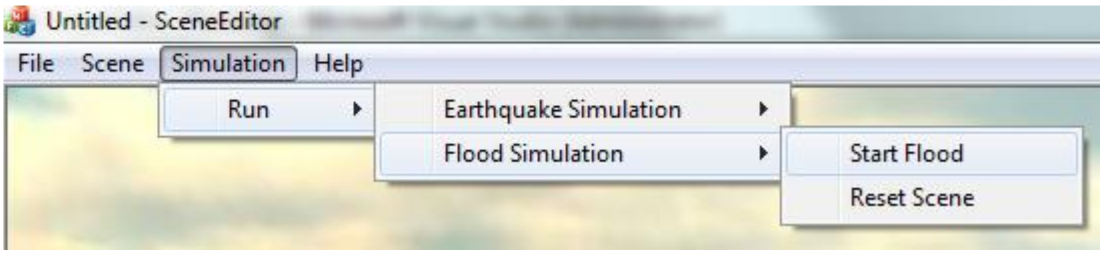
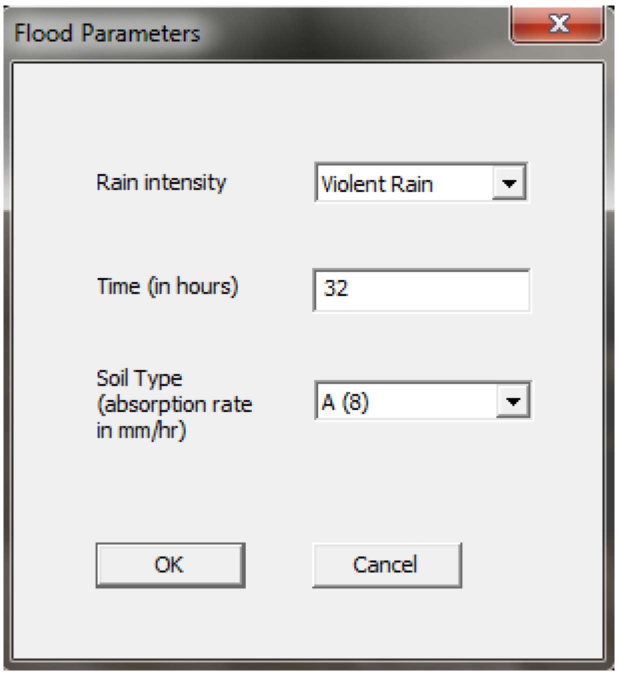


Fig 13: Starting Flood Simulation

Clicking on Start Flood opens a new dialog box as shown in figure below,


Flood Parameters

Rain intensity: Violent Rain

Time (in hours): 32

Soil Type (absorption rate in mm/hr): A (8)

Fig 14: Flood Parameter Dialog Box

For simulating flood on scenario, user has to provide several parameters. First parameter is Rain Intensity. Rain Intensity has four values viz. Light Rain, Moderate Rain, Heavy Rain and Violent Rain. Rain intensity is in mm/hour. Next parameter is Time. Time specifies the duration for rain in hours. Next parameter is Soil Type. Here user can select one soil type out of 4 soil types. Soil has been given grades according to absorption rate of the soil. A type of soil has highest absorption rate and D soil type has lowest absorption rate. Absorption rate is specified in mm/hour. Based on these parameters simulation system simulates flooding on user-constructed scenario.

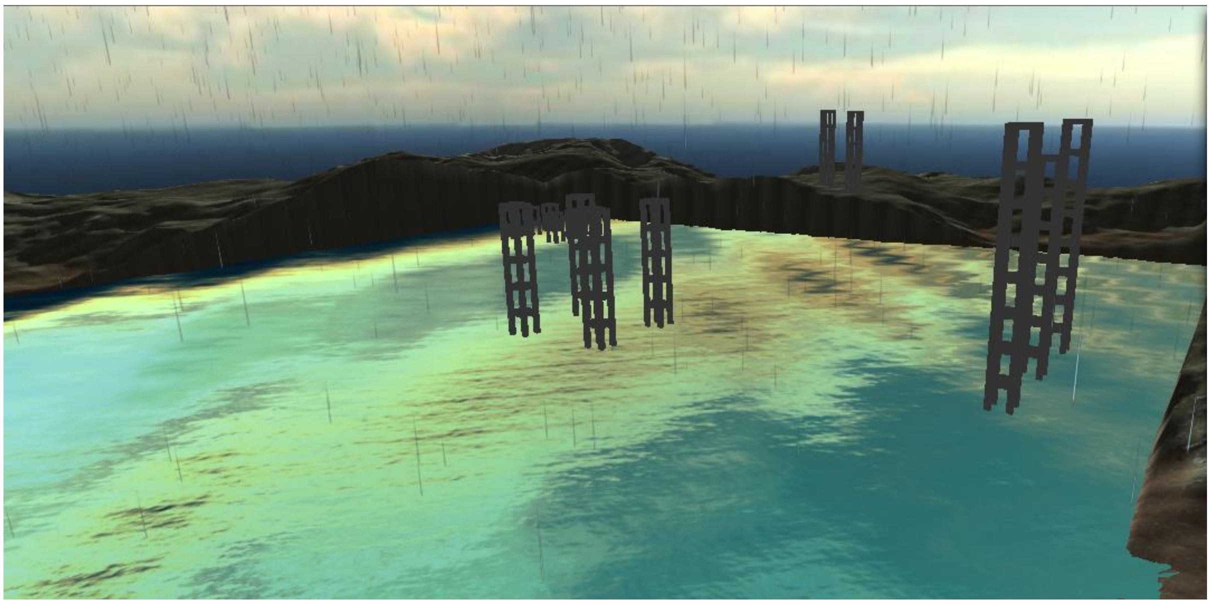
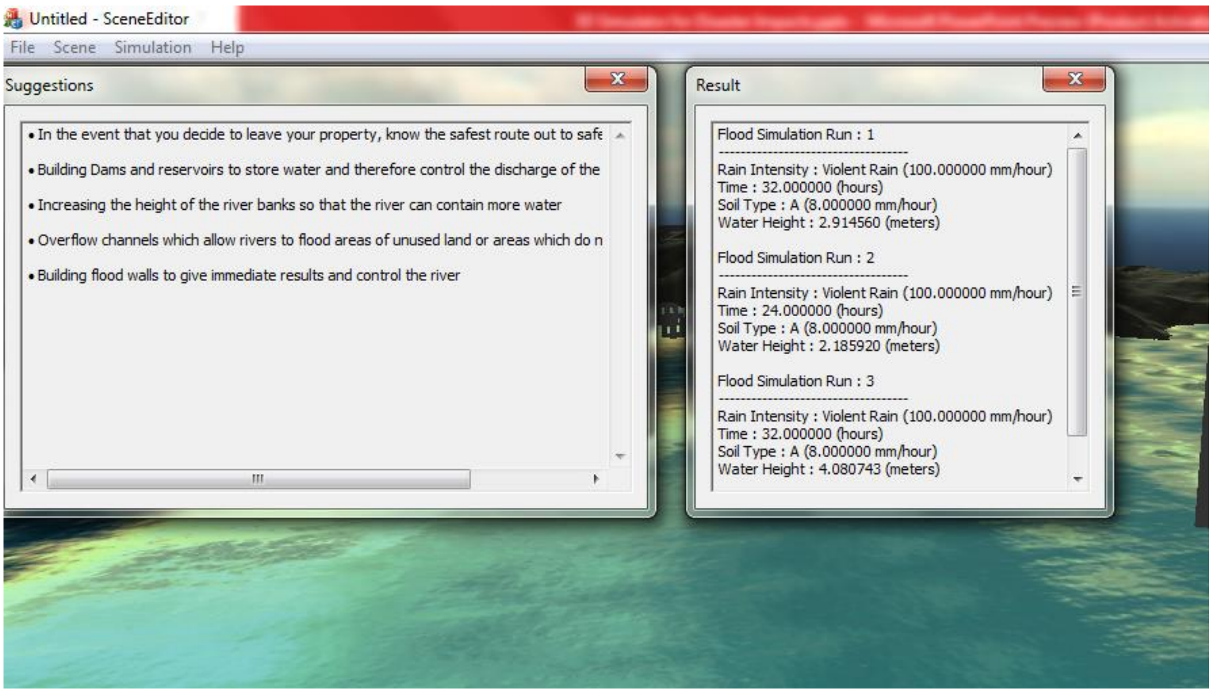


Fig 15: Flood Simulation in progress

After simulation is done, two dialog box pop-ups. First one is suggestion dialog box and second is Result dialog box.



- In the event that you decide to leave your property, know the safest route out to safe
- Building Dams and reservoirs to store water and therefore control the discharge of the
- Increasing the height of the river banks so that the river can contain more water
- Overflow channels which allow rivers to flood areas of unused land or areas which do n
- Building flood walls to give immediate results and control the river

```
Flood Simulation Run : 1
----------------------------------
Rain Intensity : Violent Rain (100.000000 mm/hour)
Time : 32.000000 (hours)
Soil Type : A (8.000000 mm/hour)
Water Height : 2.914560 (meters)

Flood Simulation Run : 2
----------------------------------
Rain Intensity : Violent Rain (100.000000 mm/hour)
Time : 24.000000 (hours)
Soil Type : A (8.000000 mm/hour)
Water Height : 2.185920 (meters)

Flood Simulation Run : 3
----------------------------------
Rain Intensity : Violent Rain (100.000000 mm/hour)
Time : 32.000000 (hours)
Soil Type : A (8.000000 mm/hour)
Water Height : 4.080743 (meters)
```

Fig 16: Flood Simulation Result

Suggestion dialog box shows the steps to stop flooding in an area and also to make it disaster proof. Result dialog box acts as a log window and keeps track of all previous simulations performed on city. So that user can compare result of simulation for different combination of parameters. If user has to run new simulation then it can be done with reset scene option from flood simulation menu.

9.3.7 Simulating Earthquake

For simulating earthquake simulation, user has to select Run option from Simulation Menu. Then click on Earthquake simulation and then on Start Simulation.

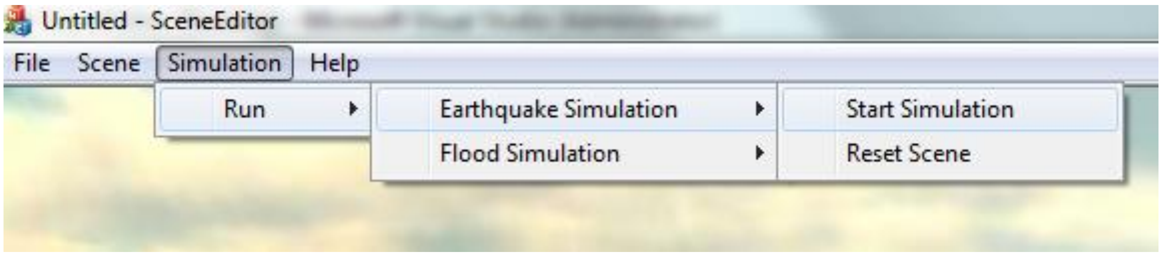
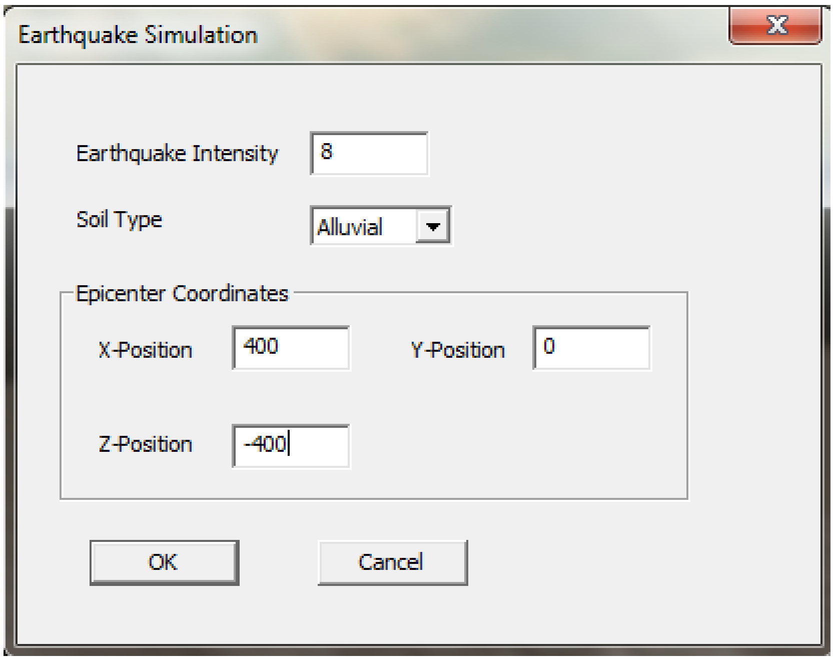


Fig 17: Starting Earthquake Simulation

Clicking on Start Simulation opens a new dialog box as shown in figure below,


Earthquake Simulation

Earthquake Intensity: 8

Soil Type: Alluvial

Epicenter Coordinates

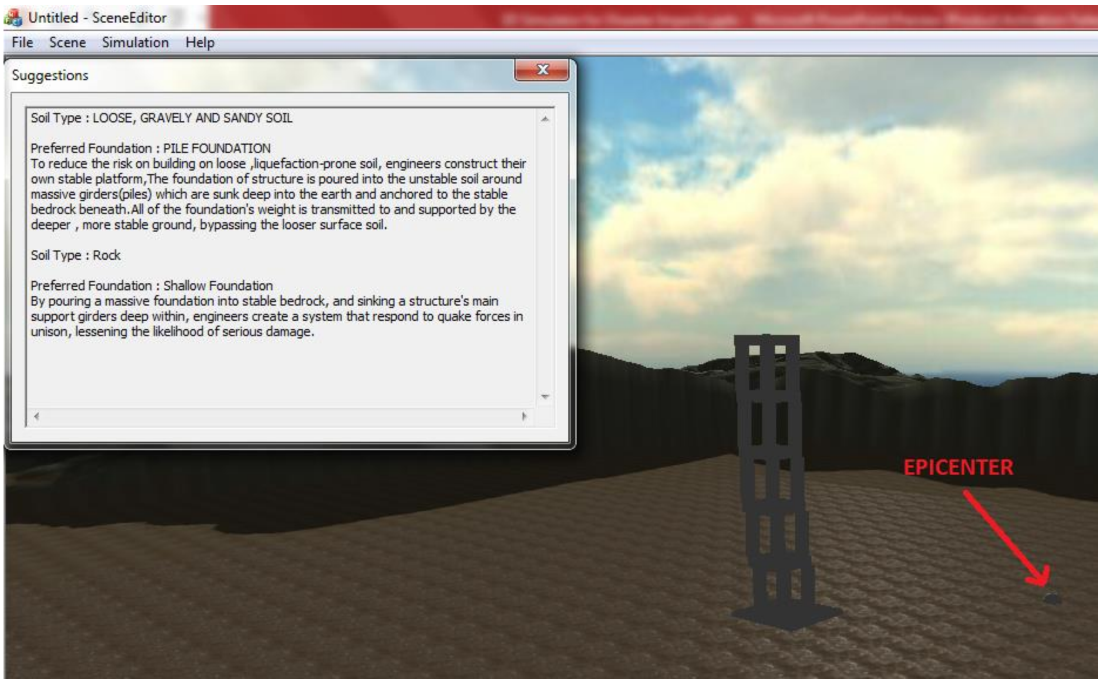
X-Position: 400

Y-Position: 0

Z-Position: -400

Fig 18: Earthquake Parameter Dialog Box

For simulation earthquake on scenario, user has to provide several parameters. First parameter is Earthquake Intensity. Earthquake Intensity is in Richter scale. Next parameter is soil type. User can select two types of soil viz. Alluvial and Rock. Next parameter is Epicenter coordinates in 3D space. Based on these parameters simulation system simulates earthquake on user constructed scenario.



Soil Type : LOOSE, GRAVELY AND SANDY SOIL

Preferred Foundation : PILE FOUNDATION
To reduce the risk on building on loose ,liquefaction-prone soil, engineers construct their own stable platform,The foundation of structure is poured into the unstable soil around massive girders(piles) which are sunk deep into the earth and anchored to the stable bedrock beneath.All of the foundation's weight is transmitted to and supported by the deeper , more stable ground, bypassing the looser surface soil.

Soil Type : Rock

Preferred Foundation : Shallow Foundation
By pouring a massive foundation into stable bedrock, and sinking a structure's main support girders deep within, engineers create a system that respond to quake forces in unison, lessening the likelihood of serious damage.

Fig 19: Earthquake Simulation Result

After simulation is done, suggestions dialog box pop-ups. It gives suggestion to make building disaster proof.

9.3.8 Loading and Saving Scene

User can save his scenario in xml file format by clicking on File and then on export scene. Similarly user can load already created scenario using import scene option from import menu.

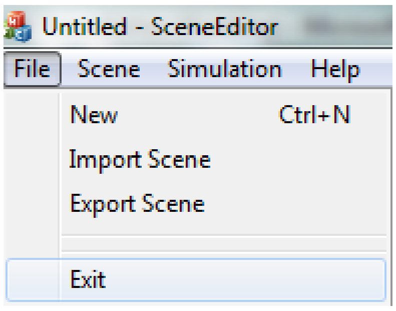
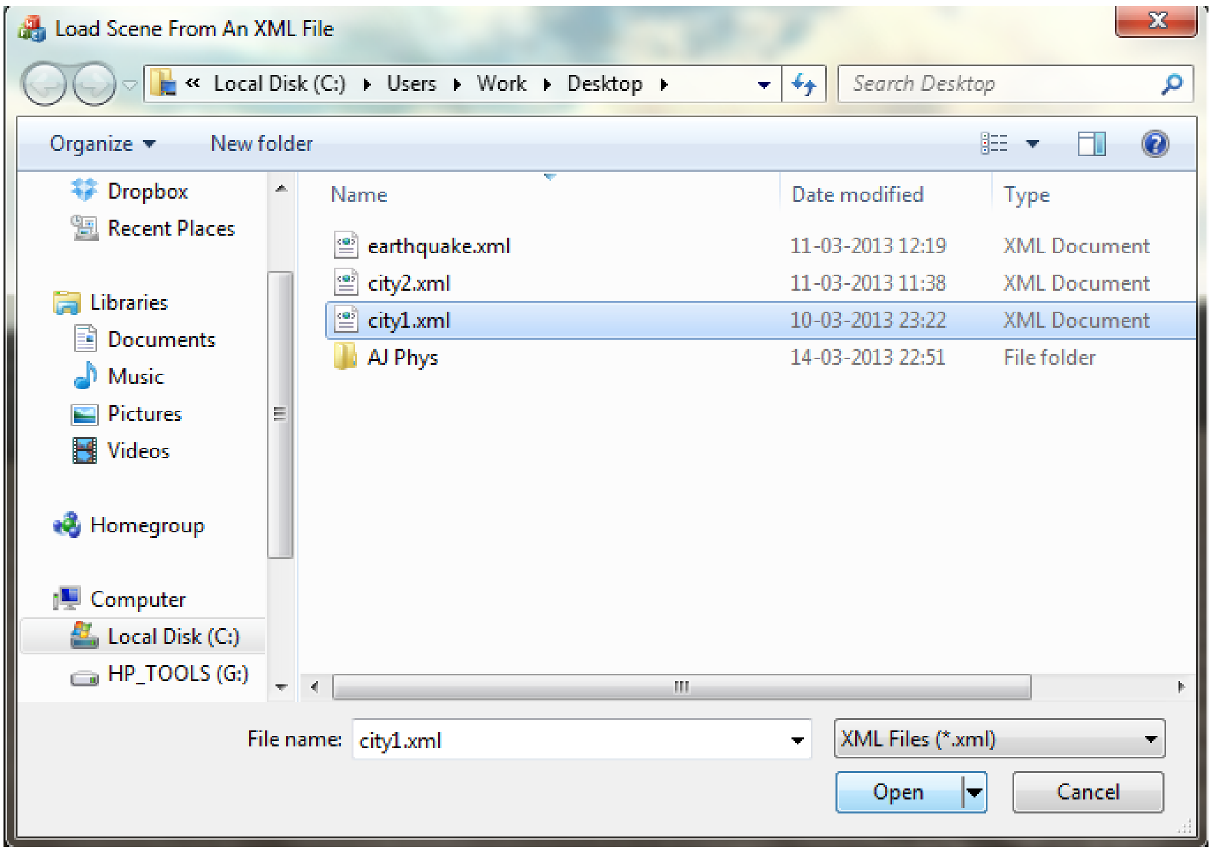


Fig 20: File Menu



Name	Date modified	Type
earthquake.xml	11-03-2013 12:19	XML Document
city2.xml	11-03-2013 11:38	XML Document
city1.xml	10-03-2013 23:22	XML Document
AJ Phys	14-03-2013 22:51	File folder

File name: city1.xml

XML Files (*.xml)

Fig 21: Loading Scene from XML File

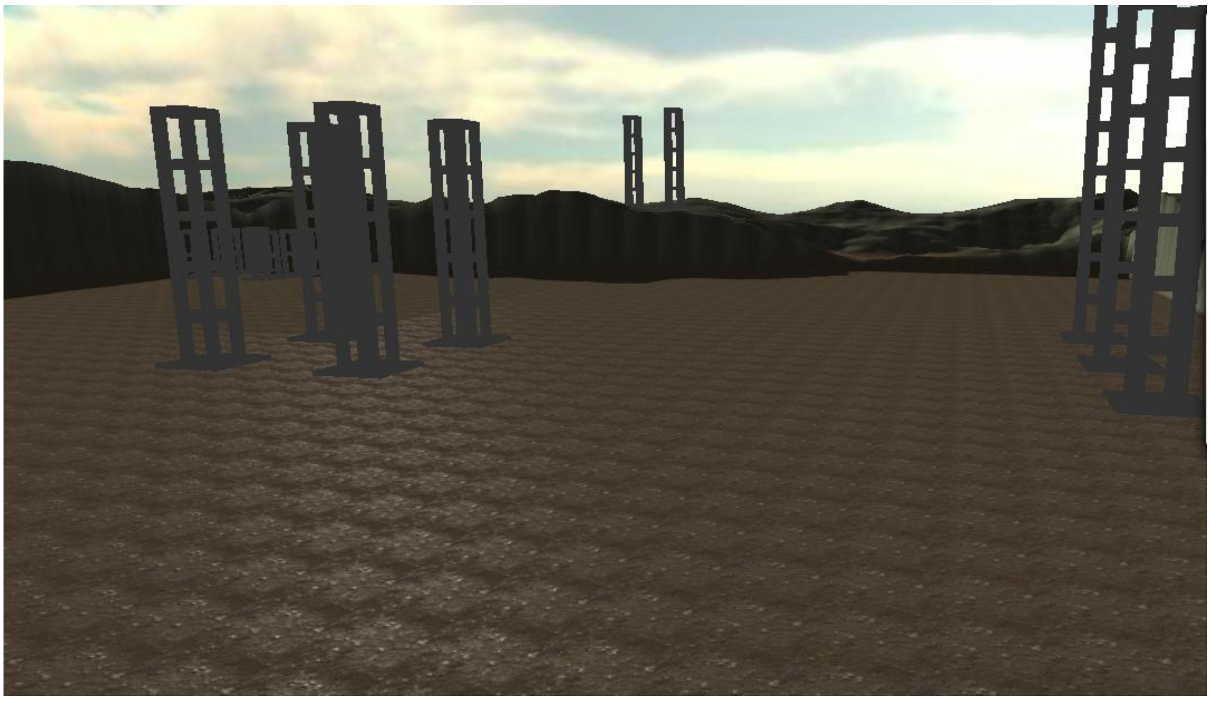


Fig 22: City loaded from XML file

10. Testing

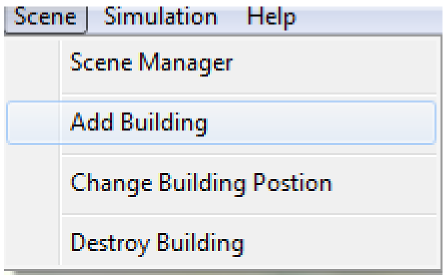
10.1 Introduction to testing

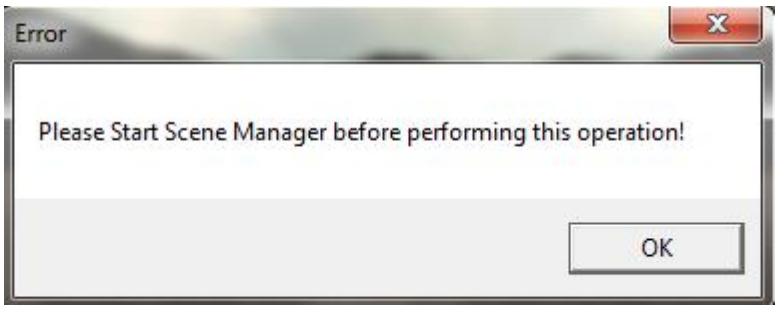
Software testing is an activity aimed at evaluating an attribute or capability of a system and determining that it meets its required results. Software is not unlike other physical processes where inputs are received and outputs are produced. The primary purpose of testing is to detect software failures so that the defects may be discovered and corrected. Information derived from software testing may be used to correct the process by which the software has been developed.

10.2 Test cases

10.2.1 Test case - Scene Manager Activation

Scene Menu operations like adding building, changing building position and deleting building requires checking of one item from scene manager dialog box. For using those operation user first needs to activate scene manger from scene menu.



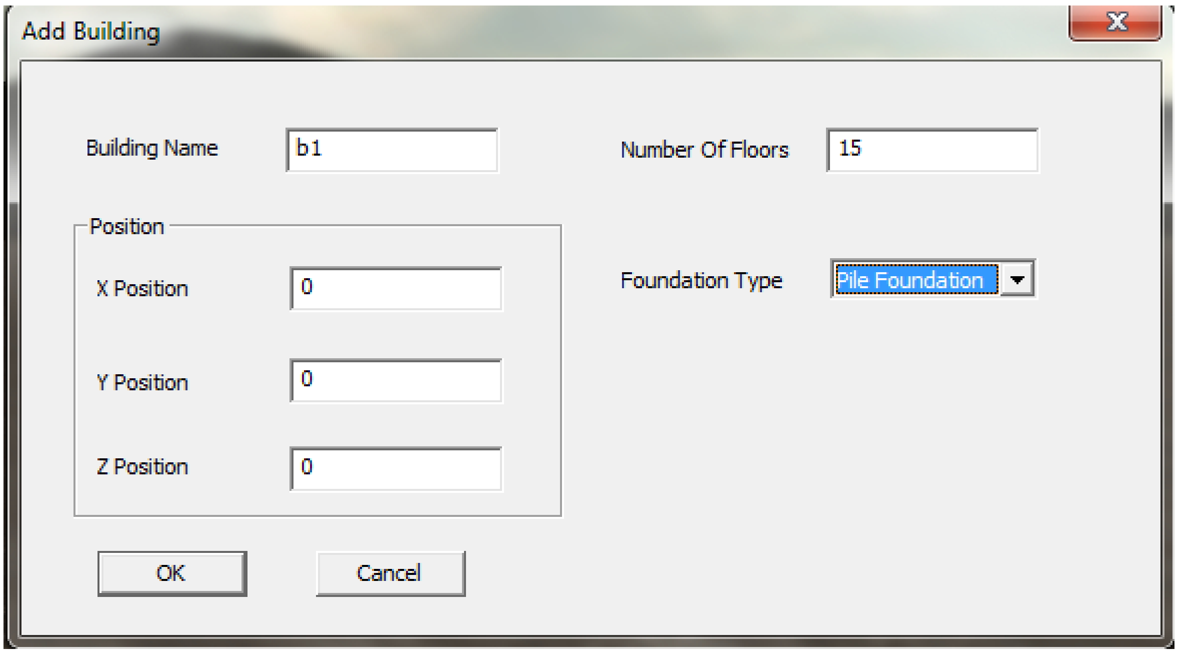


Please Start Scene Manager before performing this operation!

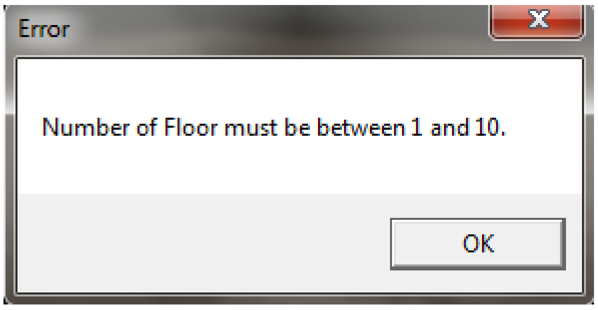
Fig 23: Test case for Scene Manager Activation

10.2.2 Test case – Building parameter validation

While creating a building user needs to provide several parameters. In this, we have constraints like number of floor of building should be in between 1 to 10. If user tries to enter number of floor greater than 10 or less than 1 then system would give error along with message so that user can understand why there was error.


Add Building

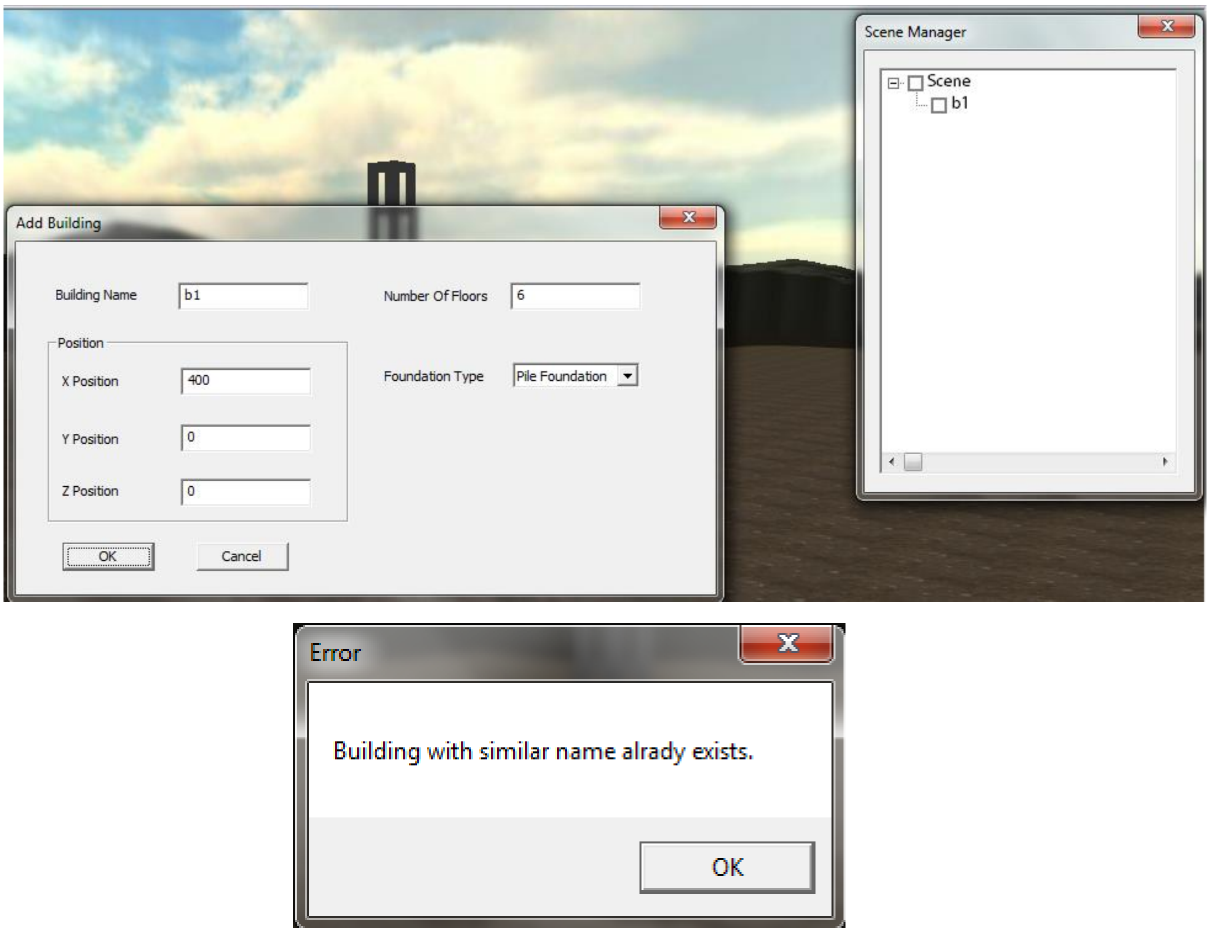
Building Name	b1	Number Of Floors	15
X Position	0	Foundation Type	Pile Foundation
Y Position	0		
Z Position	0		


Error

Number of Floor must be between 1 and 10.

Fig 24: Building parameter validation

For identification of building each should be given some name and that has to be unique name. This mechanism ensures that it will satisfy the requirement of Ogre 3D that each scenenode should have unique name.


Scene Manager

Scene
b1

Add Building

Building Name: b1
Number Of Floors: 6

Position
X Position: 400
Y Position: 0
Z Position: 0

Foundation Type: Pile Foundation

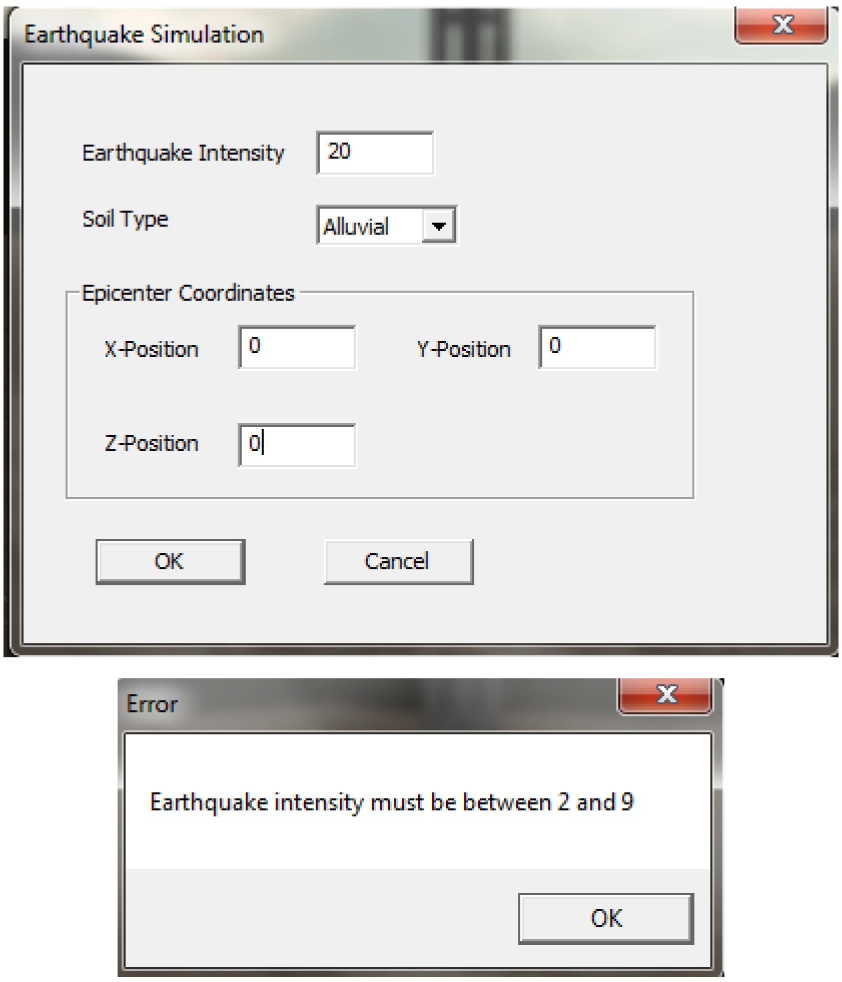
Error

Building with similar name alrady exists.

Fig 25: Building name validation

10.2.3 Test case – Earthquake intensity value validation

For earthquake simulation, user needs to provide earthquake intensity. That user intensity needs to be in between 2 and 10. If it is not in specified range then the system will give an error message.


Earthquake Simulation

Earthquake Intensity: 20

Soil Type: Alluvial

Epicenter Coordinates

X-Position: 0

Y-Position: 0

Z-Position: 0

Error

Earthquake intensity must be between 2 and 9

Fig 26: Earthquake intensity value validation

11. FUTURE SCOPE

We have provided a basic simulation of earthquake and flood. By considering more parameters for building and simulation environment, system can be made more precise and realistic.

In current system user is creating virtual city but in future we plan to integrate Geographic information system (GIS) for creating 3D scenario of real cities. Also various element trees, roads, bridge, dams etc. can be used while creating scenario.

Earthquake simulation can be extended by considering various parameters like type of material (concrete, composite, metals etc.) for creating building, dimension of building. Also we can consider different soil types with different bearing capacity. More type of foundation implemented can be. We can also consider forces applied by wind and water in case of flood.

Flood simulation can be extended by considering various parameters like slope of terrain, drainage system, and more soil types. We can also consider absorption of water by trees. Different trees has different absorption rate, so we can consider various types of trees. In case of flooding we can also provide human rescue plan for that city.

Currently then system is developed using Microsoft MFC; hence it is only available for Windows platform. In future we can port this application to Qt or WxWidget so that we can target many platforms. Also by making this project open source we can get help from many people with different areas of expertise which will help in improving quality and accuracy of simulation.

12. CONCLUSION

The intended system has been successfully implemented as a 3D simulator to simulate earthquake and flood on user constructed scenarios. The user is also allowed to import and export already created scenario saved in an xml file. The application is helpful for city planners as they can see effect of disasters on a virtual city and can take necessary steps to make disaster proof cities.

Easy to use interface and graphical simulation helps to understand and analyze the impacts of a disaster more accurately. The system will be an aid to a city planner for better planning and visualization of city layout. Widespread use and adaptation will allow the city planners to build disaster proof infrastructures. These structures will be more resistant to disasters, saving human life and property damage.

This application can be used as a base for creating simulator, which will consider more parameters of building, flood and earthquake. The implemented simulation system can also be used for other applications by simply adding or creating new mesh files for a particular application. These mesh files will give the user a freedom to create more customized scenarios, thus providing a more flexible simulation.

Our simulator can also be for educational purpose to help students understand earthquake and flood and also see effect of those disasters on a city. They can be taught about how they should act in such situation to be safe and possibly save others. Thus the simulator can be used on huge application scale and certain improvements will definitely add to the features of the system and improve its accuracy and precision.

13. REFERENCES

[1] Wikipedia contributors. (2012, October 2). Earthquake Shaking Table [Online]. Available: http://en.wikipedia.org/wiki/Earthquake_shaking_table

[2] Uno, Keisuke; Kashiyama, Kazuo, "Development of simulation system for the disaster evacuation based on multi-agent model using GIS," *Tsinghua Science and Technology*, vol.13, no.S1, pp.348, 353, Oct. 2008

[3] Sun Guangcai; Wei Wenli; Liu, Y.L., "Numerical Scheme for Simulation of 2D Flood Waves," *Computational and Information Sciences (ICCIS), 2010 International Conference on* , vol., no., pp.846,849, 17-19 Dec. 2010

[4] Taborda, R.; Bielak, J., "Large-Scale Earthquake Simulation: Computational Seismology and Complex Engineering Systems," *Computing in Science &Engineering*, vol.13, no.4, pp.14, 27, July-Aug. 2011

[5] Chourasia, A.; Cutchin, S.; Yifeng Cui; Moore, R.W.; Olsen, K.; Day, S.M.; Minster, J.B.; Maechling, P.; Jordan, T.H., "Visual Insights into High-Resolution Earthquake Simulations," *Computer Graphics and Applications, IEEE* , vol.27, no.5, pp.28,34, Sept.-Oct. 2007

[6] Xu, Feng; Chen, Xuping; Ren, Aizhu; Lu, Xinzheng, "Earthquake disaster simulation for an urban area, with GIS, CAD, FEA, and VR integration," *Tsinghua Science and Technology*, vol.13, no.S1, pp.311, 316, Oct. 2008

[7] “Prediction of peak ground acceleration during strong earthquake,”
V. K. Singh, D. K. Paul. (1994). Prediction of peak ground acceleration during strong earthquake [Online]. Available FTP: iitk.ac.in Directory: /nicee/wcee/article/
File: 6_vol1_913.pdf

14. APPENDIX

1. /* Initializing Ogre 3d and terrain creation */

```
void CSceneEditorView::EngineSetup(void)
{
        Ogre::Root *Root = ((CSceneEditorApp*)AfxGetApp())->m_Engine-
>GetRoot();
        m_SceneManager = Root->createSceneManager(Ogre::ST_GENERIC,
"SceneEditor");

Ogre::NameValuePairList parms;
parms["externalWindowHandle"] = Ogre::StringConverter::toString((long)m_hWnd);
parms["vsync"] = "true";

CRect   rect;
GetClientRect(&rect);
        Ogre::RenderTarget *RenderWindow = Root-
>getRenderTarget("SceneEditor");

        if (RenderWindow == NULL)
        {
        try
        {
                m_RenderWindow = Root->createRenderWindow("SceneEditor",
rect.Width(), rect.Height(), false, &parms);
        }
catch(...)
        {
                MessageBox("Cannot initialize\nCheck that graphic-card driver is up-to-
date", "Initialize Render System", MB_OK | MB_ICONSTOP);
                exit(EXIT_SUCCESS);
        }
        }
        Ogre::ResourceGroupManager::getSingleton().initialiseAllResourceGroups();
        m_Camera = m_SceneManager->createCamera("Camera");
        m_Camera->setPosition(Ogre::Vector3(1683, 50, 2116));
m_Camera->lookAt(Ogre::Vector3(0, 0, 0));
m_Camera->setNearClipDistance(0.1);
m_Camera->setFarClipDistance(50000);

if (m_Root->getRenderSystem()->getCapabilities()-
>hasCapability(Ogre::RSC_INFINITE_FAR_PLANE))
   {
m_Camera->setFarClipDistance(0);   // enable infinite far clip distance if we can
   }

        m_Camera->setAutoTracking(false);
        m_Camera->setFixedYawAxis(true);
```

```
	m_Camera->yaw(Ogre::Radian(0));
	m_Camera->pitch(Ogre::Radian(0));

	Ogre::SceneNode *CameraNode = NULL;
	CameraNode = m_SceneManager->getRootSceneNode()-
>createChildSceneNode("CameraNode");

	//Create viewport
	Ogre::Viewport* Viewport = NULL;

	if (0 == m_RenderWindow->getNumViewports())
	{
		Viewport = m_RenderWindow->addViewport(m_Camera);
		Viewport->setBackgroundColour(Ogre::ColourValue(0.0f, 0.0f, 0.0f));
	}

m_Camera->setAspectRatio(Ogre::Real(rect.Width()) / Ogre::Real(rect.Height()));

	//setup frameListener
	createFrameListener();

	Ogre::MaterialManager::getSingleton().setDefaultTextureFiltering(Ogre::TFO_
ANISOTROPIC);
   Ogre::MaterialManager::getSingleton().setDefaultAnisotropy(7);

	m_SceneManager->setSkyBox(true, "Examples/CloudyNoonSkyBox");

	m_SceneManager->setFog(Ogre::FOG_LINEAR, Ogre::ColourValue(0.7, 0.7,
0.8), 0, 10000, 25000);
	Ogre::Vector3 lightdir(0.55, -0.3, 0.75);
	lightdir.normalise();

	Ogre::Light* l = m_SceneManager->createLight("Light1");
	l->setType(Ogre::Light::LT_DIRECTIONAL);
	l->setDirection(lightdir);
	l->setDiffuseColour(Ogre::ColourValue::White);
	l->setSpecularColour(Ogre::ColourValue(0.4, 0.4, 0.4));

	m_SceneManager->setAmbientLight(Ogre::ColourValue(0.2, 0.2, 0.2));

	mTerrainGlobals = OGRE_NEW Ogre::TerrainGlobalOptions();
	mTerrainGroup = OGRE_NEW Ogre::TerrainGroup(m_SceneManager,
Ogre::Terrain::ALIGN_X_Z, 513, 12000.0f);
mTerrainGroup->setFilenameConvention(Ogre::String("Terrain"), Ogre::String("dat"));
mTerrainGroup->setOrigin(Ogre::Vector3::ZERO);

configureTerrainDefaults(l);

for (long x = 0; x <= 0; ++x)
for (long y = 0; y <= 0; ++y)
```

```
defineTerrain(x, y);

mTerrainGroup->loadAllTerrains(true);

if (mTerrainsImported)
   {
       Ogre::TerrainGroup::TerrainIterator ti = mTerrainGroup->getTerrainIterator();
while(ti.hasMoreElements())
       {
           Ogre::Terrain* t = ti.getNext()->instance;
initBlendMaps(t);
       }
   }

mTerrainGroup->freeTemporaryResources();

           createPillarMesh();
           createSlabMesh();
           createFloorMesh();

           //import and export
           bname=new CString[20];
           foundation_type = new int[20];
           floors=new int[20];
           xpos=new int[20];
           ypos=new int[20];
           zpos=new int[20];
           counter_pos = new int[20];

           Globals::dbgdraw = new BtOgre::DebugDrawer(m_SceneManager-
>getRootSceneNode(), Globals::phyWorld);
           Globals::phyWorld->setDebugDrawer(Globals::dbgdraw);

           Ogre::Terrain *pTerrain = mTerrainGroup->getTerrain(0,0);
           float* terrainHeightData = pTerrain->getHeightData();
   Ogre::Vector3 terrainPosition = pTerrain->getPosition();

float * pDataConvert= new float[pTerrain->getSize() *pTerrain->getSize()];
for(int i=0;i<pTerrain->getSize();i++)
memcpy(
pDataConvert+pTerrain->getSize() * i, // source
terrainHeightData + pTerrain->getSize() * (pTerrain->getSize()-i-1), // target
sizeof(float)*(pTerrain->getSize()) // size
            );
           float metersBetweenVertices = pTerrain->getWorldSize()/(pTerrain->getSize()-
1);
btVector3 localScaling(metersBetweenVertices, 1, metersBetweenVertices);

btHeightfieldTerrainShape* terrainShape = new btHeightfieldTerrainShape(
pTerrain->getSize(),
pTerrain->getSize(),
pDataConvert,
      1,
pTerrain->getMinHeight(),
pTerrain->getMaxHeight(),
      1,
      PHY_FLOAT,
true);

         terrainShape->setUseZigzagSubdivision(true);
terrainShape->setLocalScaling(localScaling);

btRigidBody *mTerrainBody = new btRigidBody(0, new btDefaultMotionState(),
terrainShape);

mTerrainBody->getWorldTransform().setOrigin(
btVector3(
               terrainPosition.x,
terrainPosition.y + (pTerrain->getMaxHeight()-pTerrain->getMinHeight())/2,
               terrainPosition.z));

mTerrainBody->getWorldTransform().setRotation(
btQuaternion(
               Ogre::Quaternion::IDENTITY.x,
               Ogre::Quaternion::IDENTITY.y,
               Ogre::Quaternion::IDENTITY.z,
               Ogre::Quaternion::IDENTITY.w));
         mTerrainBody->setFriction(10);
          Globals::phyWorld->addRigidBody(mTerrainBody);
}
```

2. /*Creating pillar, slab mesh*/

```
void CSceneEditorView::createPillarMesh()
{
                  Ogre::ManualObject* lManualObject = NULL;
                  {
                           Ogre::String lManualObjectName = "pillar";
                           lManualObject = m_SceneManager-
>createManualObject(lManualObjectName);

                           bool lDoIWantToUpdateItLater = false;
                           lManualObject->setDynamic(lDoIWantToUpdateItLater);

                           lManualObject->begin("",
Ogre::RenderOperation::OT_TRIANGLE_LIST);
                           {
                                    float cp = 10.0f;
                                    float cm = -10.0f;
```

```
                    lManualObject->position(cm, 8*cp, -10);// a vertex
                    lManualObject->position(cp, 8*cp, -10);// a vertex
                    lManualObject->position(cp, cm, -10);// a vertex
                    lManualObject->position(cm, cm, -10);// a vertex

                    lManualObject->position(cm, 8*cp, 10);// a vertex
                    lManualObject->position(cp, 8*cp, 10);// a vertex
                    lManualObject->position(cp, cm, 10);// a vertex
                    lManualObject->position(cm, cm, 10);// a vertex

                    lManualObject->triangle(0,1,2);
                    lManualObject->triangle(2,3,0);
                    lManualObject->triangle(4,6,5);
                    lManualObject->triangle(6,4,7);

                    lManualObject->triangle(0,4,5);
                    lManualObject->triangle(5,1,0);
                    lManualObject->triangle(2,6,7);
                    lManualObject->triangle(7,3,2);

                    lManualObject->triangle(0,7,4);
                    lManualObject->triangle(7,0,3);
                    lManualObject->triangle(1,5,6);
                    lManualObject->triangle(6,2,1);
               }
               lManualObject->end();
          }
          Ogre::String lNameOfTheMesh = "pillar";
          Ogre::String lResourceGroup = 
Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME;
          lManualObject->convertToMesh(lNameOfTheMesh);

}

void CSceneEditorView::createSlabMesh()
{
          Ogre::ManualObject* lManualObject = NULL;
          {
               Ogre::String lManualObjectName = "slab";
               lManualObject = m_SceneManager-
>createManualObject(lManualObjectName);

               bool lDoIWantToUpdateItLater = false;
               lManualObject->setDynamic(lDoIWantToUpdateItLater);

               lManualObject->begin("", 
Ogre::RenderOperation::OT_TRIANGLE_LIST);
               {
                    float cp = 10.0f;
                    float cm = -10.0f;
```

```
					lManualObject->position(cm, cp, -10);// a vertex
					lManualObject->position(8*cp, cp, -10);// a vertex
					lManualObject->position(8*cp, cm, -10);// a vertex
					lManualObject->position(cm, cm, -10);// a vertex

					lManualObject->position(cm, cp, 80);// a vertex
					lManualObject->position(8*cp, cp, 80);// a vertex
					lManualObject->position(8*cp, cm, 80);// a vertex
					lManualObject->position(cm, cm, 80);// a vertex

					lManualObject->triangle(0,1,2);
					lManualObject->triangle(2,3,0);
					lManualObject->triangle(4,6,5);
					lManualObject->triangle(6,4,7);

					lManualObject->triangle(0,4,5);
					lManualObject->triangle(5,1,0);
					lManualObject->triangle(2,6,7);
					lManualObject->triangle(7,3,2);

					lManualObject->triangle(0,7,4);
					lManualObject->triangle(7,0,3);
					lManualObject->triangle(1,5,6);
					lManualObject->triangle(6,2,1);
				}
				lManualObject->end();
		}
		Ogre::String lNameOfTheMesh = "slab";
		Ogre::String lResourceGroup =
Ogre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME;
		lManualObject->convertToMesh(lNameOfTheMesh);
}
```

3. /*Adding Building code */

```
void CSceneEditorView::OnSceneAddbuilding()
{
	// TODO: Add your command handler code here
	if(isSceneMgrStarted)
	{
		CAddBuilding addBuilding;
		if (IDOK == addBuilding.DoModal())
		{
			HTREEITEM Selected = m_SceneManagerDlg-
>m_SceneTree.GetSelectedItem();
			Ogre::String name = m_SceneManagerDlg-
>m_SceneTree.GetItemText(Selected);
			bool isnodePresent = m_SceneManager->hasSceneNode(name);
			if(name == "Scene" && !isnodePresent)
```

```
                        {
                                if(!addBuilding.name.IsEmpty())
                                {
                                        if(!m_SceneManager-
>hasSceneNode(Ogre::String(addBuilding.name)))
                                        {
                                                if(addBuilding.num > 0 &&
addBuilding.num < 11)
                                                {
                                                        m_SceneManagerDlg-
>m_SceneTree.InsertItem(addBuilding.name, Selected);
                                                        m_SceneManagerDlg-
>m_SceneTree.Expand(Selected, TVE_EXPAND);

        createBuilding(Ogre::String(addBuilding.name),
addBuilding.mFoundationType_Val,addBuilding.num, atoi(addBuilding.x),
atoi(addBuilding.y), atoi(addBuilding.z));
                                                }
                                                else
                                                {
                                                        MessageBox("Number of Floor
must be between 1 and 10.","Error",0);
                                                }
                                        }
                                        else
                                        {
                                                MessageBox("Building with similar name
alrady exists.","Error",0);
                                        }
                                }
                                else
                                {
                                        MessageBox("Building name can not be
empty.","Error",0);
                                }
                        }
                        else
                        {
                                MessageBox("Scene not seleted!","Error",0);
                        }
                }
        }
        else
        {
                MessageBox("Please Start Scene Manager before performing this
operation!","Error",0);
        }
}
```

```
void CSceneEditorView::createBuilding(Ogre::String name,int foundation, int floor,
long x,long y,long z)
{
	//Store the data to be exported to xml
	bname[index]=name.c_str ();
	foundation_type[index] = foundation;
	floors[index]=floor;
	xpos[index]=x;
	ypos[index]=y;
	zpos[index]=z;
	counter_pos[index] = counter;
	index++;

	Ogre::SceneNode *building_node = m_SceneManager->getRootSceneNode()-
>createChildSceneNode(name,Ogre::Vector3(x,y,z));

	Ogre::SceneNode *ground_node = building_node-
>createChildSceneNode(Ogre::String("ground_node_"+Ogre::StringConverter::toString
(counter)),Ogre::Vector3(-30,0,-30));
	Ogre::Entity *ground_entity = m_SceneManager-
>createEntity(Ogre::String("ground_entity_"+Ogre::StringConverter::toString(counter))
, "ground");
	ground_node->attachObject(ground_entity);
	counter++;

	for(int k=0; k<floor; k++)
	{
		Ogre::SceneNode *pillar_node_1 = building_node-
>createChildSceneNode(Ogre::String("pillar_node_"+Ogre::StringConverter::toString(
counter)),((Ogre::Vector3(0,20,0))+k*Ogre::Vector3(0,110,0)));
		Ogre::Entity *pillar_entity_1 = m_SceneManager-
>createEntity(Ogre::String("pillar_entity_"+Ogre::StringConverter::toString(counter)),
"pillar");
		pillar_node_1->attachObject(pillar_entity_1);
		counter++;

		Ogre::SceneNode *pillar_node_2 = building_node-
>createChildSceneNode(Ogre::String("pillar_node_"+Ogre::StringConverter::toString(
counter)), (Ogre::Vector3(70,20,0)+k*Ogre::Vector3(0,110,0)));
		Ogre::Entity *pillar_entity_2 = m_SceneManager-
>createEntity(Ogre::String("pillar_entity_"+Ogre::StringConverter::toString(counter)),
"pillar");
		pillar_node_2->attachObject(pillar_entity_2);
		counter++;

		Ogre::SceneNode *pillar_node_3 = building_node-
>createChildSceneNode(Ogre::String("pillar_node_"+Ogre::StringConverter::toString(
counter)), (Ogre::Vector3(0,20,70)+k*Ogre::Vector3(0,110,0)));
```

```
			Ogre::Entity *pillar_entity_3 = m_SceneManager-
>createEntity(Ogre::String("pillar_entity_"+Ogre::StringConverter::toString(counter)),
"pillar");
			pillar_node_3->attachObject(pillar_entity_3);
			counter++;

			Ogre::SceneNode *pillar_node_4 = building_node-
>createChildSceneNode(Ogre::String("pillar_node_"+Ogre::StringConverter::toString(
counter)), (Ogre::Vector3(70,20,70)+k*Ogre::Vector3(0,110,0)));
			Ogre::Entity *pillar_entity_4 = m_SceneManager-
>createEntity(Ogre::String("pillar_entity_"+Ogre::StringConverter::toString(counter)),
"pillar");
			pillar_node_4->attachObject(pillar_entity_4);
			counter++;

			Ogre::SceneNode *slab_node = building_node-
>createChildSceneNode(Ogre::String("slab_node_"+Ogre::StringConverter::toString(c
ounter)), (Ogre::Vector3(0,110,0)+k*Ogre::Vector3(0,110,0)));
			Ogre::Entity *slab_entity = m_SceneManager-
>createEntity(Ogre::String("slab_entity_"+Ogre::StringConverter::toString(counter)),"s
lab");
			slab_node->attachObject(slab_entity);
			counter++;
		}
}
```

3. /*keyboard and mouse navigation*/

```
void CSceneEditorView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{
	//KillTimer(m_WorkingTimer);
	KillTimer(m_WorkingTimer);
	m_WorkingTimer = 0;

	switch (nChar)
	{
		case VK_LEFT: //left
			m_WorkingTimer = 1;
			break;
		case VK_UP:  //up
			m_WorkingTimer = 2;
			break;
		case VK_RIGHT: //right
			m_WorkingTimer = 3;
			break;
		case VK_DOWN: //down
			m_WorkingTimer = 4;
			break;
	}
	if (m_WorkingTimer != 0)
	SetTimer(m_WorkingTimer, 10, NULL);
	CView::OnKeyDown(nChar, nRepCnt, nFlags);
}

void CSceneEditorView::OnKeyUp(UINT nChar, UINT nRepCnt, UINT nFlags)
{
	KillTimer(m_WorkingTimer);
	CView::OnKeyUp(nChar, nRepCnt, nFlags);
}

void CSceneEditorView::OnLButtonDown(UINT nFlags, CPoint point)
{
	m_MousePosition = point;
	m_MouseNavigation = true;
	CView::OnLButtonDown(nFlags, point);
}

BOOL CSceneEditorView::OnMouseWheel(UINT nFlags, short zDelta, CPoint pt)
{
	Ogre::Vector3 CameraMove(0.0, 0.0, 0.0);
	CameraMove[2] = 1 * zDelta;
	CEngine * Engine = ((CSceneEditorApp*)AfxGetApp())->m_Engine;
	if (Engine == NULL)
		return false;
	Ogre::Root *Root = Engine->GetRoot();
	if (m_Camera == NULL)
		return false;
	m_Camera->moveRelative(CameraMove);
	return CView::OnMouseWheel(nFlags, zDelta, pt);
}

void CSceneEditorView::OnMouseMove(UINT nFlags, CPoint point)
{
	if (m_MouseNavigation)
	{
		Ogre::Vector3 CameraMove(0.0, 0.0, 0.0);
		CameraMove[0] = -(m_MousePosition.x - point.x);
		CameraMove[1] = -(m_MousePosition.y - point.y);
		CEngine * Engine = ((CSceneEditorApp*)AfxGetApp())->m_Engine;
		if (Engine == NULL)
			return;
		Ogre::Root *Root = Engine->GetRoot();
		if (m_Camera == NULL)
			return;
		m_Camera->yaw(Ogre::Degree(CameraMove[0] * 0.10f));
		m_Camera->pitch(Ogre::Degree(CameraMove[1] * 0.10f));
		m_MousePosition = point;
	}
	CView::OnMouseMove(nFlags, point);
}
```

```
void CSceneEditorView::OnTimer(UINT_PTR nIDEvent)
{
	Ogre::Vector3 CameraMove;
	Ogre::Root *Root = ((CSceneEditorApp*)AfxGetApp())->m_Engine-
>GetRoot();

	switch (nIDEvent)
	{
		case 1: CameraMove[0] = -10;
			CameraMove[1] = 0;
			CameraMove[2] = 0;
			m_Camera->moveRelative(CameraMove);
			break;
		case 2: CameraMove[0] = 0;
			CameraMove[1] = 10;
			CameraMove[2] = 0;
			m_Camera->moveRelative(CameraMove);
			break;
		case 3: CameraMove[0] = 10;
			CameraMove[1] = 0;
			CameraMove[2] = 0;
			m_Camera->moveRelative(CameraMove);
			break;
		case 4: CameraMove[0] = 0;
			CameraMove[1] = -10;
			CameraMove[2] = 0;
			m_Camera->moveRelative(CameraMove);
			break;
		}
}
```

4. /*water and rain creation code */

```
void CSceneEditorView::createWater()
{
mWaterNode = m_SceneManager->getRootSceneNode()-
>createChildSceneNode("WaterNode");
Ogre::Plane waterSurface;
waterSurface.normal = Ogre::Vector3::UNIT_Y;
waterSurface.d = 20;
Ogre::MeshManager::getSingleton().createPlane("WaterSurface",
gre::ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
waterSurface,12000, 12000, 50, 50, true, 1, 1, 1, Ogre::Vector3::UNIT_Z);

mWaterSurfaceEnt = m_SceneManager->createEntity( "WaterSurface",waterSurface");
mWaterNode->attachObject(mWaterSurfaceEnt);
mWaterNode->setPosition(Ogre::Vector3(0,0,0));
mWaterSurfaceEnt->setMaterialName("Ocean2_HLSL_GLSL");
}
```

```
void CSceneEditorView::createRain()
{
	Ogre::ParticleSystem::setDefaultNonVisibleUpdateTimeout(5);
	Ogre::ParticleSystem* ps;
	ps = m_SceneManager->createParticleSystem("Rain", "Examples/Rain");
ps->fastForward(5);
	m_SceneManager->getRootSceneNode()-
>createChildSceneNode("RainNode",Ogre::Vector3(0, 1000, 0))->attachObject(ps);
}
```

5. /*loading and saving scene*/

```
void CSceneEditorView::OnFileImportScene()
{
CFileDialog dlg(TRUE);
dlg.m_ofn.lpstrFilter=_T("XML Files (*.xml)\0*.xml\0All Files (*.*)\0*.*\0\0");
dlg.m_ofn.lpstrTitle=_T("Load Scene From An XML File");
 CString filename;

if(dlg.DoModal() == IDOK)
{  filename = dlg.GetPathName();
		clearScene();
		import_xml.Load(filename);
		import_xml.FindElem(_T("count"));
		int count= _ttoi(import_xml.GetData());

		if(count!=0)
		{   import_xml.ResetPos ();
			import_xml.FindElem(_T("buildings"));
			import_xml.IntoElem ();
			for(int i=0;i<count;i++)
			{
				import_xml.FindElem(_T("building"));
				import_xml.IntoElem ();
				import_xml.FindElem(_T("name"));
				bname[i]=import_xml.GetData();
				import_xml.FindElem(_T("floors"));
				floors[i]=_ttoi( import_xml.GetData());
				import_xml.FindElem(_T("foundation"));
				foundation_type[i]=_ttoi( import_xml.GetData());
				import_xml.FindElem(_T("xpos"));
				xpos[i]=_ttoi( import_xml.GetData());
				import_xml.FindElem(_T("ypos"));
				ypos[i]= _ttoi(import_xml.GetData());
				import_xml.FindElem(_T("zpos"));
				zpos[i]=_ttoi(import_xml.GetData());
				import_xml.OutOfElem();
			}
			for(int i=0;i<count;i++)
			{
```

```
                                        HTREEITEM Selected = m_SceneManagerDlg-
>m_SceneTree.GetSelectedItem();
                                        Ogre::String name = m_SceneManagerDlg-
>m_SceneTree.GetItemText(Selected);
                                        m_SceneManagerDlg-
>m_SceneTree.InsertItem(bname[i], Selected);
                                        m_SceneManagerDlg->m_SceneTree.Expand(Selected,
TVE_EXPAND);

          createBuilding(Ogre::String(bname[i]),foundation_type[i],floors[i],xpos[i],ypos[
i],zpos[i]);
                              }
                              MessageBox("Import Successfull!","Import",MB_OK);
                    }
          }
}

void CSceneEditorView::OnFileExportScene()
{
          if(index!=0)
          {
                    CFileDialog dlgsave(FALSE);
                    dlgsave.m_ofn.lpstrFilter=_T("XML Files (*.xml)\0*.xml\0Text Files
(*.txt)\0*.txt\0All Files (*.*)\0*.*\0\0");
                    dlgsave.m_ofn .lpstrDefExt =_T(".xml");
                    dlgsave.m_ofn.lpstrTitle=_T("Save Scene As XML File");

                    CString filename;
                    if(dlgsave.DoModal() == IDOK)
                    {
                              filename = dlgsave.GetPathName();
                              export_xml=NULL;

                              export_xml.AddNode (1,_T("buildings"));

                              for(int i=0;i<index;i++)
                              {
                                        export_xml.FindElem(_T("buildings"));
                                        export_xml.IntoElem();
                                        export_xml.AddNode(1,_T("building"));
                                        export_xml.FindElem(_T("building"));
                                        export_xml.IntoElem();
                                        export_xml.AddElem(_T("name"),_T(bname[i]),0);
                    export_xml.AddElem(_T("foundation"),_T(foundation_type[i]),0);
                                        export_xml.AddElem(_T("floors"),floors[i],0);
                                        export_xml.AddElem(_T("xpos"),(int)xpos[i],0);
                                        export_xml.AddElem(_T("ypos"),(int)ypos[i],0);
                                        export_xml.AddElem(_T("zpos"),(int)zpos[i],0);
                                        export_xml.OutOfElem();
                                        export_xml.OutOfElem();
```

```
                        }
                        export_xml.AddElem (_T("count"),index,0);
                        export_xml.Save(filename);
                        MessageBox("Export Successfull!","Export",MB_OK);
                }
        }
}
```

6.   /*Earthquake simulation code */

```
void CSceneEditorView::OnRunEsim()
{
        CEarthParamDlg paramDlg;
        if(IDOK == paramDlg.DoModal())
        {
                if(index >= 1)
                {
                        float intensity = atof(paramDlg.intensity);
                        if(intensity > 1.9 && intensity < 9.1)
                        {
                                for(int i=0;i<index;i++)
                                {
                                        //get building
                                        Ogre::SceneNode *building_node =
m_SceneManager->getSceneNode(Ogre::String(bname[i]));
                                        building_node-
>setPosition(Ogre::Vector3(0,0,0));
                                        int c = counter_pos[i];
                                        Ogre::Vector3 pos(xpos[i],ypos[i],zpos[i]);
                                        //get floor
                                        Ogre::SceneNode *ground_node =
(Ogre::SceneNode *)building_node-
>getChild(Ogre::String("ground_node_"+Ogre::StringConverter::toString(c)));
                                        Ogre::Entity *ground_entity = m_SceneManager-
>getEntity(Ogre::String("ground_entity_"+Ogre::StringConverter::toString(c)));
                                        c++;
                                        ground_node->setPosition(pos+ground_node-
>getPosition());
                                        //Create the ground shape.
                                        BtOgre::StaticMeshToShapeConverter
converter1(ground_entity);
                                        mGroundShape = converter1.createTrimesh();

                                        //Create BtOgre MotionState (connects Ogre and
Bullet).
                                        btDefaultMotionState *groundState = new
btDefaultMotionState(btTransform(btQuaternion(0,0,0,1),BtOgre::Convert::toBullet(gr
ound_node->_getDerivedPosition())));
                                        //Create the Body.
                                        mGroundBody[i] = new btRigidBody(0,
groundState, mGroundShape, btVector3(0,0,0));
                                        Globals::phyWorld-
>addRigidBody(mGroundBody[i]);
                                        //mass of pillar
                                        btScalar mass = 5;
                                        btVector3 inertia;
                                        int bodyCounter=0;
                                        for(int k=0; k<floors[i]; k++)
                                        {
                                                //first pillar
                                                Ogre::SceneNode *pillar_node_1 =
(Ogre::SceneNode *)building_node-
>getChild(Ogre::String("pillar_node_"+Ogre::StringConverter::toString(c)));
                                                Ogre::Entity *pillar_entity_1 =
m_SceneManager-
>getEntity(Ogre::String("pillar_entity_"+Ogre::StringConverter::toString(c)));
                                                c++;

                                                //Create shape.
                                                BtOgre::StaticMeshToShapeConverter
converter2(pillar_entity_1);
                                                mPillarShape =
converter2.createConvex();
                                                //Calculate inertia.
                                                mPillarShape->calculateLocalInertia(mass,
inertia);
                                                //Create BtOgre MotionState (connects
Ogre and Bullet).
                                                BtOgre::RigidBodyState* mPillarState =
new
BtOgre::RigidBodyState(pillar_node_1,btTransform(btQuaternion(0,0,0,1),BtOgre::Co
nvert::toBullet(pos+pillar_node_1->getPosition())));
                                                mPillarState;
                                                //Create the Body.
                                                mBody[i][bodyCounter] = new
btRigidBody(mass, mPillarState, mPillarShape, inertia);
                                                Globals::phyWorld-
>addRigidBody(mBody[i][bodyCounter]);
                                                bodyCounter++;

                                                //second pillar
                                                Ogre::SceneNode *pillar_node_2 =
(Ogre::SceneNode *)building_node-
>getChild(Ogre::String("pillar_node_"+Ogre::StringConverter::toString(c)));
                                                Ogre::Entity *pillar_entity_2 =
m_SceneManager-
>getEntity(Ogre::String("pillar_entity_"+Ogre::StringConverter::toString(c)));
                                                c++;
                                                //Create shape.
                                                BtOgre::StaticMeshToShapeConverter
converter3(pillar_entity_2);
                                                mPillarShape =
converter3.createConvex();
                                                //Calculate inertia.
                                                mPillarShape->calculateLocalInertia(mass,
inertia);
                                                //Create BtOgre MotionState (connects
Ogre and Bullet).
                                                mPillarState = new
BtOgre::RigidBodyState(pillar_node_2,btTransform(btQuaternion(0,0,0,1),BtOgre::Co
nvert::toBullet(pos+pillar_node_2->getPosition())));
                                                //Create the Body.
                                                mBody[i][bodyCounter] = new
btRigidBody(mass, mPillarState, mPillarShape, inertia);
                                                Globals::phyWorld-
>addRigidBody(mBody[i][bodyCounter]);
                                                bodyCounter++;

                                                //third pillar
                                                Ogre::SceneNode *pillar_node_3 =
(Ogre::SceneNode *)building_node-
>getChild(Ogre::String("pillar_node_"+Ogre::StringConverter::toString(c)));
                                                Ogre::Entity *pillar_entity_3 =
m_SceneManager-
>getEntity(Ogre::String("pillar_entity_"+Ogre::StringConverter::toString(c)));
                                                c++;

                                                //Create shape.
                                                BtOgre::StaticMeshToShapeConverter
converter4(pillar_entity_3);
                                                mPillarShape =
converter4.createConvex();
                                                //Calculate inertia.
                                                mPillarShape->calculateLocalInertia(mass,
inertia);
                                                //Create BtOgre MotionState (connects
Ogre and Bullet).
                                                mPillarState = new
BtOgre::RigidBodyState(pillar_node_3,btTransform(btQuaternion(0,0,0,1),BtOgre::Co
nvert::toBullet(pos+pillar_node_3->getPosition())));
                                                //Create the Body.
                                                mBody[i][bodyCounter] = new
btRigidBody(mass, mPillarState, mPillarShape, inertia);
                                                Globals::phyWorld-
>addRigidBody(mBody[i][bodyCounter]);
                                                bodyCounter++;

                                                //fourth pillar
```

```
                                        Ogre::SceneNode *pillar_node_4 =
(Ogre::SceneNode *)building_node-
>getChild(Ogre::String("pillar_node_"+Ogre::StringConverter::toString(c)));
                                        Ogre::Entity *pillar_entity_4 =
m_SceneManager-
>getEntity(Ogre::String("pillar_entity_"+Ogre::StringConverter::toString(c)));
                                        c++;

                                        //Create shape.
                                        BtOgre::StaticMeshToShapeConverter
converter5(pillar_entity_4);
                                        mPillarShape =
converter5.createConvex();
                                        //Calculate inertia.
                                        mPillarShape->calculateLocalInertia(mass,
inertia);
                                        //Create BtOgre MotionState (connects
Ogre and Bullet).
                                        mPillarState = new
BtOgre::RigidBodyState(pillar_node_4,btTransform(btQuaternion(0,0,0,1),BtOgre::Co
nvert::toBullet(pos+pillar_node_4->getPosition())));
                                        //Create the Body.
                                        mBody[i][bodyCounter] = new
btRigidBody(mass, mPillarState, mPillarShape, inertia);
                                        Globals::phyWorld-
>addRigidBody(mBody[i][bodyCounter]);
                                        bodyCounter++;

                                        //slab
                                        Ogre::SceneNode *slab_node =
(Ogre::SceneNode *)building_node-
>getChild(Ogre::String("slab_node_"+Ogre::StringConverter::toString(c)));
                                        Ogre::Entity *slab_entity =
m_SceneManager-
>getEntity(Ogre::String("slab_entity_"+Ogre::StringConverter::toString(c)));
                                        c++;

                                        //Create shape.
                                        BtOgre::StaticMeshToShapeConverter
converter(slab_entity);
                                        mSlabShape = converter.createConvex();
                                        //Calculate inertia.
                                        btScalar mass1 = 10;
                                        btVector3 inertia1;
                                        mSlabShape-
>calculateLocalInertia(mass1, inertia1);
                                        //Create BtOgre MotionState (connects
Ogre and Bullet).
                                        BtOgre::RigidBodyState* mSlabState =
new
```

```
BtOgre::RigidBodyState(slab_node,btTransform(btQuaternion(0,0,0,1),BtOgre::Conver
t::toBullet(pos+slab_node->getPosition())));
                                                //Create the Body.
                                                mBody[i][bodyCounter] = new
btRigidBody(mass, mSlabState, mSlabShape, inertia1);
                                                Globals::phyWorld-
>addRigidBody(mBody[i][bodyCounter]);
                                                bodyCounter++;
                                        }
                                }

                                int *dist = new int[index];
                                int *iarray = new int[index];
                                int *findex = new int[index];
                                Ogre::Vector3 epicenter;
                                epicenter.x = atoi(paramDlg.x);
                                epicenter.y = atoi(paramDlg.y);
                                epicenter.z = atoi(paramDlg.z);
                                Ogre::SceneNode *epi_sphere = m_SceneManager-
>getRootSceneNode()->createChildSceneNode("epicenter",epicenter);
                                Ogre::Entity * epi_entity = m_SceneManager-
>createEntity("epi","sphere.mesh");
                                epi_sphere->attachObject(epi_entity);
                                epi_sphere->scale(0.2,0.2,0.2);
                                //calculate distance between building and epicenter and
store it in dist[] array
                                for(int j=0;j<index;j++)
                                {
                                        Ogre::Vector3 b_pos;
                                        b_pos.x = xpos[j];
                                        b_pos.y = ypos[j];
                                        b_pos.z = zpos[j];
                                        dist[j] = epicenter.distance(b_pos);
                                        iarray[j] = j;
                                        findex[j] = foundation_type[j];
                                }
                                //sort building bodies according to distance from the
epicenter
                                for (int i = 0 ; i < ( index - 1 ); i++)
                                {
                                        for (int j = 0 ; j < index - i - 1; j++)
                                        {
                                                if (dist[j] > dist[j+1])
                                                {       //swap distance
                                                        int swap1 = dist[j];
                                                        dist[j]   = dist[j+1];
                                                        dist[j+1] = swap1;
                                                        //swap building index
                                                        int swap2 = iarray[j];
                                                        iarray[j] = iarray[j+1];
                                                        iarray[j+1] = swap2;
                                                        //swap building foundations
                                                        int swap3 = findex[j];
                                                        findex[j] = findex[j+1];
                                                        findex[j+1] = swap3;
                                                }
                                        }
                                }

                                int sType = paramDlg.soilType_val;
                                float accel;
                                float factor;
                                if(sType == 0)
                                {
                                        for(int i=0;i<index;i++)
                                        {       //foundation
                                                int val = findex[i];
                                                if(val == 0)
                                                        factor = 0.3;
                                                else if(val == 1)
                                                        factor = 0.6;
                                                else
                                                        factor = 1;

                                                accel = 33 * factor * 0.0135 *
(pow(dist[i],-1.37)) * (pow(10, (0.5 * intensity)));
                                                float force = accel * floors[iarray[i]]*50;

                                                for(int m=0;m<50;m++)
                                                {
                                                        if(mBody[i][m] != NULL)
                                                        {

                                                                if(epicenter.x -
xpos[iarray[i]] > 0)
                                                                {

                        mBody[iarray[i]][m]->setActivationState(DISABLE_DEACTIVATION);

                        mBody[iarray[i]][m]->applyCentralForce(btVector3(-force,0,0));
                                                                }else if(epicenter.x -
xpos[iarray[i]] < 0)
                                                                {

                        mBody[iarray[i]][m]->setActivationState(DISABLE_DEACTIVATION);

                        mBody[iarray[i]][m]->applyCentralForce(btVector3(force,0,0));
                                                                }
                                                                if(epicenter.z -
zpos[iarray[i]] > 0)
```

```
                                                    {

        mBody[iarray[i]][m]->setActivationState(DISABLE_DEACTIVATION);

        mBody[iarray[i]][m]->applyCentralForce(btVector3(0,0,-force));
                                                    }else if(epicenter.z -
zpos[iarray[i]] < 0)
                                                    {

        mBody[iarray[i]][m]->setActivationState(DISABLE_DEACTIVATION);

        mBody[iarray[i]][m]->applyCentralForce(btVector3(0,0,force));
                                                    }
                                            }
                                    }
                            }
                    }
                    if(sType == 1)
                    {
                            for(int i=0;i<index;i++)
                            {       //foundation
                                    if(findex[i] == 0)
                                            factor = 1;
                                    else if(findex[i] == 1)
                                            factor = 0.6;
                                    else
                                            factor = 0.3;

                                    accel = 33 * factor * 0.0135 *
(pow(dist[i],-1.4)) * (pow(10, (0.5 * intensity)));
                                    float force = accel * floors[iarray[i]]*50;

                                    for(int m=0;m<50;m++)
                                    {
                                            if(mBody[i][m] != NULL)
                                            {
                                                    if(epicenter.x -
xpos[iarray[i]] > 0)
                                                    {

        mBody[iarray[i]][m]->setActivationState(DISABLE_DEACTIVATION);

        mBody[iarray[i]][m]->applyCentralForce(btVector3(-force,0,0));
                                                    }else if(epicenter.x -
xpos[iarray[i]] < 0)
                                                    {

        mBody[iarray[i]][m]->setActivationState(DISABLE_DEACTIVATION);

        mBody[iarray[i]][m]->applyCentralForce(btVector3(force,0,0));
```

```
                                                }
                                                if(epicenter.z -
zpos[iarray[i]] > 0)
                                                {

    mBody[iarray[i]][m]->setActivationState(DISABLE_DEACTIVATION);

    mBody[iarray[i]][m]->applyCentralForce(btVector3(0,0,-force));
                                                }else if(epicenter.z -
zpos[iarray[i]] < 0)
                                                {

    mBody[iarray[i]][m]->setActivationState(DISABLE_DEACTIVATION);

    mBody[iarray[i]][m]->applyCentralForce(btVector3(0,0,force));
                                                }
                                            }
                                        }
                                    }
                            }
                            if(this->m_suggestionDlg == NULL)
                            {
                                m_suggestionDlg = new CSuggestions();
                                m_suggestionDlg->Create(IDD_S_D);
                            }
                            CString sug_str1 = "Soil Type : LOOSE, GRAVELY
AND SANDY SOIL\r\n";
                            CString     sug_str2 = "Preferred Foundation : PILE
FOUNDATION\r\nTo reduce the risk on building on loose ,liquefaction-prone soil,
engineers construct their\r\nown stable platform,The foundation of structure is poured
into the unstable soil around\r\nmassive girders(piles) which are sunk deep into the
earth and anchored to the stable\r\nbedrock beneath.All of the foundation's weight is
transmitted to and supported by the\r\ndeeper , more stable ground, bypassing the looser
surface soil.\r\n";
                            CString sug_str3 = "Soil Type : Rock\r\n";
                            CString     sug_str4 = "Preferred Foundation :
Shallow Foundation\r\nBy pouring a massive foundation into stable bedrock, and
sinking a structure's main\r\nsupport girders deep within, engineers create a system that
respond to quake forces in\r\nunison, lessening the likelihood of serious damage.";
                            m_suggestionDlg->sug_edit.SetWindowText(sug_str1 +
"\r\n"+sug_str2 + "\r\n"+sug_str3+ "\r\n"+sug_str4);
                            m_suggestionDlg->UpdateWindow();
                            m_suggestionDlg->ShowWindow(SW_SHOW);
                        }
                        else
                        {
                            MessageBoxA("Earthquake intensity must be between 2
and 9","Error",0);
                        }
                }
```

```
		else
		{
			MessageBoxA("No Buildings present to simulate 
earthtquake!","Error",0);
		}
	}
```

7. /*Flood simulation code */

```
void CSceneEditorView::OnFloodsimulationStartflood()
{
	// TODO: Add your command handler code her
	CFloodParamerterDlg floodParameterDlg;

	if(IDOK == floodParameterDlg.DoModal())
	{
		currTime = floodParameterDlg.time;
		float diff;
		if(floodParameterDlg.mSoilType_val == 0) //Soli type A
		{
			soilType = "A";
			currAbsorptionRate = 8; //in mm
			if(floodParameterDlg.mRainIntensity_val == 0) //Light Rain
			{
				rainType = "Light Rain";
				currRainIntensity = 2.5; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}

			if(floodParameterDlg.mRainIntensity_val == 1) //Moderate Rain
			{
				rainType = "Moderate Rain";
				currRainIntensity = 10; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}

			if(floodParameterDlg.mRainIntensity_val == 2) //heavy Rain
			{
				rainType = "Heavy Rain";
				currRainIntensity = 50; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}

			if(floodParameterDlg.mRainIntensity_val == 3) //Violent Rain
			{
				rainType = "Violent Rain";
				currRainIntensity = 100; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}
		}

		if(floodParameterDlg.mSoilType_val == 1) //Soli type B
		{
			soilType = "B";
			currAbsorptionRate = 4; //in mm
			if(floodParameterDlg.mRainIntensity_val == 0) //Light Rain
			{
				rainType = "Light Rain";
				currRainIntensity = 2.5; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}

			if(floodParameterDlg.mRainIntensity_val == 1) //Moderate Rain
			{
				rainType = "Moderate Rain";
				currRainIntensity = 10; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}

			if(floodParameterDlg.mRainIntensity_val == 2) //heavy Rain
			{
				rainType = "Heavy Rain";
				currRainIntensity = 50; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}

			if(floodParameterDlg.mRainIntensity_val == 3) //Violent Rain
			{
				rainType = "Violent Rain";
				currRainIntensity = 100; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}
		}

		if(floodParameterDlg.mSoilType_val == 2) //Soli type C
		{
			soilType = "C";
			currAbsorptionRate = 1.3; //in mm
			if(floodParameterDlg.mRainIntensity_val == 0) //Light Rain
			{
				rainType = "Light Rain";
				currRainIntensity = 2.5; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}

			if(floodParameterDlg.mRainIntensity_val == 1) //Moderate Rain
			{
				rainType = "Moderate Rain";
				currRainIntensity = 10; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
				}
			}

			if(floodParameterDlg.mRainIntensity_val == 2) //heavy Rain
			{
				rainType = "Heavy Rain";
				currRainIntensity = 50; //in mm
				diff = currRainIntensity - currAbsorptionRate;
				if(diff > 0)
				{
					currWaterHeight = diff * currTime;
```

```
                }
            }

            if(floodParameterDlg.mRainIntensity_val == 3) //Violent Rain
            {
                rainType = "Violent Rain";
                currRainIntensity = 100; //in mm
                diff = currRainIntensity - currAbsorptionRate;
                if(diff > 0)
                {
                    currWaterHeight = diff * currTime;
                }
            }
        }

        if(floodParameterDlg.mSoilType_val == 3) //Soli type D
        {
            soilType = "D";
            currAbsorptionRate = 0.7; //in mm
            if(floodParameterDlg.mRainIntensity_val == 0) //Light Rain
            {
                rainType = "Light Rain";
                currRainIntensity = 2.5; //in mm
                diff = currRainIntensity - currAbsorptionRate;
                if(diff > 0)
                {
                    currWaterHeight = diff * currTime;
                }
            }

            if(floodParameterDlg.mRainIntensity_val == 1) //Moderate Rain
            {
                rainType = "Moderate Rain";
                currRainIntensity = 10; //in mm
                diff = currRainIntensity - currAbsorptionRate;
                if(diff > 0)
                {
                    currWaterHeight = diff * currTime;
                }
            }

            if(floodParameterDlg.mRainIntensity_val == 2) //heavy Rain
            {
                rainType = "Heavy Rain";
                currRainIntensity = 50; //in mm
                diff = currRainIntensity - currAbsorptionRate;
                if(diff > 0)
                {
                    currWaterHeight = diff * currTime;
                }
                }

                if(floodParameterDlg.mRainIntensity_val == 3) //Violent Rain
                {
                        rainType = "Violent Rain";
                        currRainIntensity = 100; //in mm
                        diff = currRainIntensity - currAbsorptionRate;
                        if(diff > 0)
                        {
                                currWaterHeight = diff * currTime;
                        }
                }
        }
        currWaterHeight = 0.03 * currWaterHeight; //mm to pixel conversion
        additionWater();

        createRain();
        createWater();
        SetTimer(5,1,0);
    }
}
```