ABC-UTC Guideline for:

PREDICTIVE COMPUTER PROGRAM FOR PROACTIVE DEMOLITION PLANNING

May 2019

End Date:
May 31, 2019

Performing Institution:
Florida International University

Principal Investigator:
Seung Jae Lee

Graduate Research Assistant:
Ali Bakhtiari
ABSTRACT
This guideline is developed in reference to the research activities undertaken through an ABC-UTC project – Predictive Computer Program for Proactive Demolition Planning – and presents the tutorial such that engineers can leverage the computer programs adopted in this project to proactively plan the bridge demolition. The information will be of interest to highway officials, bridge (de)constructors, and structural engineers, as well as others concerned with the safe and cost-effective bridge demolition.

A considerable number of bridges in the United States are in urgent need of replacing or rehabilitation due to rapid deterioration, for which the entire or partial demolition of the existing structure is the first step to be done. While maintaining the integrity of the neighboring infrastructure and safety of workers are of high importance during the demolition process, detailed guidelines or specifications are not available for engineers and contractors for a proactive demolition planning. The potential risks and failure are hard to estimate due to inherent uncertainties in the various modes of demolition. Computer simulations to realistically predict the demolition scenarios may be useful, which enables to potentially control or possibly eliminate the hazards and inefficiency ahead. This study aims to enhance the predictive capabilities by developing a numerical simulation framework that can be useful for the engineers and contractors to realistically model, simulate and visualize the bridge demolition, which in turn better supports the project decision making.

Building upon the discrete mechanical simulation technique with high fidelity, this project aims to realistically simulate the interaction between the bridge and the demolition equipment or the explosive blast waves. Therefore, the software requires to numerically model the displacement of debris where the material transition from a continuous medium to multiple broken discontinuous pieces is considered as well as the corresponding deformation of damaged bridge. To this end, discrete element modeling is adopted, and the impulse-based simulation framework is considered to optimize the performance between computing cost and simulation fidelity. The simulations are validated using available data and video records obtained from previous demolition projects. This project also introduces a computational simulation add-on entitled as Bridge Demolition Add-on (v. 1.0.0) which is developed as an add-on to Blender, which is a free and open-source 3D computer graphics software that uses the impulse-based Bullet, a physics library which is also free and open-source.
ACKNOWLEDGMENTS

The research resulted in development of this guideline was supported by the US Department of Transportation through the Accelerated Bridge Construction University Transportation Center (ABC-UTC). The opinions, findings, or conclusions expressed in this article are solely those of the authors and may not represent the opinion of the sponsor.
CONTENTS

ABSTRACT .................................................................................................................. I

CONTENTS .................................................................................................................. III

LIST OF FIGURES ....................................................................................................... V

1. INTRODUCTION ...................................................................................................... 8
   1.1. Background ........................................................................................................ 8
   1.2. Scope of the guideline ...................................................................................... 8
   1.3. Intended users .................................................................................................. 9

2. MODELING, SIMULATION AND VISUALIZATION .................................................. 10
   2.1. Preprocessing .................................................................................................. 10
       2.1.1. Geometry modeling .................................................................................. 10
       2.1.2. Discretization and fracturing .................................................................... 18
       2.1.3. Building constraints .................................................................................. 22
       2.1.4. Sequence modeling ................................................................................... 24
       2.1.5. Rendering .................................................................................................. 25
   2.2. Bridge Demolition Add-on ............................................................................... 26
       2.2.1. Overview ................................................................................................... 26
       2.2.2. BDA installation ....................................................................................... 27
       2.2.3. User interface and features ....................................................................... 29
       2.2.4. Contact force calculation and contour visualization ................................. 32
       2.2.5. Implementation of capability to simulate the demolition by explosion ...... 33
   2.3. How to use Bridge Demolition Add-On ............................................................. 45
   2.4. Calibration, verification and examples .............................................................. 50

3. PHYSICS IMPLEMENTATION WITH BULLET ....................................................... 60
   3.1. Bullet physics library ....................................................................................... 60
   3.2. Data types and math library ............................................................................ 60
   3.3. Collision detection ........................................................................................... 60
   3.4. Rigid body dynamics ....................................................................................... 64
   3.5. Motion update .................................................................................................. 65
   3.6. Constraints modeling ....................................................................................... 66
3.7. Setting global variables ........................................................................................................ 65
3.8. Time integration ................................................................................................................ 67

SELECTION GUIDE / PROCEDURE ......................................................................................... 68

4. REFERENCES .......................................................................................................................... 71

5. APPENDIX A: IMPLEMENTED CODES ............................................................... ERROR! BOOKMARK NOT DEFINED.
LIST OF FIGURES

Figure 1. Modeling a 2-span bridge in Blender ................................................................. 11
Figure 2. Cube, a primitive mesh shown in 1) solid shading and 2) wireframe shading [1] ...... 11
Figure 3. Transform tab .................................................................................................. 12
Figure 4. Intersecting faces [1] ....................................................................................... 13
Figure 5. Boolean modifier to extract overlapping vertices, edges and faces in a compound object ................................................................................................................. 13
Figure 6. Difference, Union and Intersect in Boolean Modifier to create a single compound shape [1] .................................................................................................................. 14
Figure 7. Edit mode select mode buttons [1] ..................................................................... 14
Figure 8. A sphere in Edit mode with all of vertices selected. Edges and faces can be selected separately .................................................................................................................. 14
Figure 9. A sphere in Edit mode with some faces selected .................................................... 15
Figure 10. A modified geometry by operations in Edit mode ................................................. 15
Figure 11. Modifiers menu [1] .......................................................................................... 16
Figure 12. Rigid body parameters and options in Rigid Body modifier ....................... 17
Figure 13. Collapse of the bridge model due to pre-fracturing and not selecting start deactivated option .................................................................................................................. 18
Figure 14. Enabling cell fracture add-on in Blender’s user preference. ............................ 18
Figure 15. Cell Fracture add-on enabled in the Tools bar. ............................................... 19
Figure 16. Cell Fracture features and parameters .............................................................. 19
Figure 17. Discretized bridge by using cell fracture add-on ............................................. 20
Figure 18. Physic Body Tools bar for assigning rigid body properties .............................. 20
Figure 19. Discretization of a 2-span bridge model by Fracture modifier ...................... 21
Figure 20. Fracture settings ............................................................................................. 21
Figure 21. Fracture Constraint settings ............................................................................ 22
Figure 22. Constraint breaking settings .......................................................................... 22
Figure 23. Adding Bullet Viewport Constraints Tool ..................................................... 22
Figure 24. Bullet Viewport Constraints Tool user interface ........................................... 23
Figure 25. Constraints created ......................................................................................... 24
Figure 26. Animated option for considering animated movement of a rigid body in a simulation ........................................................................................................................................ 24
Figure 27. Blender Dope Sheet, setting keyframes for a specific action ............................ 25
Figure 28. Providing light to the rendering scene by big emission planes ....................... 25
Figure 58. The one of the remaining decks of I-235 Bridge demolished by the wrecking ball ... 55
Figure 59. The two span I-235 Bridge modeled ................................................................. 56
Figure 60. Simulated demolition of the one of remaining decks in I-235 Bridge ................. 56
Figure 61. Simulated demolition of the three span I-235 Bridge, where the wrecking ball is dropped near the end of the bridge ................................................................................. 57
Figure 62. Simulated demolition of the three span I-235 Bridge, where the wrecking ball is dropped near the center of the neighboring bridge deck ......................................................... 57
Figure 63. Simulated demolition of the three span I-235 Bridge, where the wrecking ball is dropped on the mid bridge deck; the hitting point is between the center of the mid bridge deck and the neighboring deck on the left in the figure ............................................................................. 58
Figure 64. Simulated demolition of the three span I-235 Bridge, where the wrecking ball is dropped at the center of mid bridge ............................................................................................. 58
Figure 65. Dismantling demolition using a jackhammer ...................................................... 59
Figure 66. Simulated demolition by jackhammer .................................................................. 59
Figure 68. Most used bounding shapes. [11] .................................................................... 61
Figure 69. AABBs used for Broad Phase. [12] .................................................................... 61
Figure 70. Broad Phase using AABBs [11] ........................................................................ 61
Figure 71. Sort and sweep algorithm [11] .......................................................................... 62
Figure 72. Confinement of a convex shape by a convex hull [11]. Error! Bookmark not defined.
Figure 73. SAT algorithm [11] .......................................................................................... 63
Figure 74. Bullet Physics Manual flowchart for selecting appropriate collision shape. [10] .... 64
Figure 75. Point to point constraint [10] ............................................................................. 64
Figure 76. Hinge constraint [10] ......................................................................................... 64
Figure 77. Slider constraint [10] ........................................................................................ 64
1. INTRODUCTION

1.1. Background

Bridges represent a significant subpopulation of our civil infrastructure. Majority of them are deteriorating fast and in need of replacement or rehabilitation. The first step of those replacement/rehabilitation projects is typically to either entirely or partly demolish the existing structure. Therefore, proactive planning for controlled demolition is of utmost importance to proceed with the rest of construction project in a timely manner. Maintaining the integrity of neighboring infrastructure (e.g., permanent roadways, nearby transmission lines) and the safety of workers are critical issues, for which contingency plans also must be developed based on any feasible emergency scenarios.

However, little effort has been given to develop better removal techniques of existing structures, while great effort has been made to design/construction techniques for new structures. Planning failure is often unpredictably realized in the demolition project due to inherent uncertainty hard to characterize ahead, not only in the deteriorated condition of the structure that may be far different from that of the original design, but also in the mode of destruction that may depend on adopted demolition methods, types and performance of destruction tools, dismantling sequence and associated change in the remaining structural capacity during the demolition process.

It is typically hard to develop a general guideline/specification that can facilitate safe and efficient demolition, and very limited information has been available to guide structural engineers and contractors on how to proceed with the demolition of an existing structure. This lack of generalized procedure has led to structural engineers and contractors approaching the demolition work differently, and as a result, most states neither specify parameters for demolition equipment nor require the submission of contractor qualifications with the demolition plans [1], [2].

1.2. Scope of the guideline

This guideline summarizes the research activities undertaken through an ABC-UTC project – Predictive Computer Program for Proactive Demolition Planning. The features of software package are introduced in this guideline including Bridge Demolition Add-on (BDA) v 1.0.0 that has been developed as an add-on to Blender [3], a free and open-source 3D computer graphics software that uses the impulse-based Bullet [4] engine, a physics library which is also free and open-source. The programming of BDA is done in Python using Blender’s API to create and develop the add-on,
where additional features are implemented such as contact force retrieval that are not available in the existing software. This guideline provides the tutorial such that engineers can leverage the software package to proactively plan the bridge demolition project. The simulation is demonstrated for the demolition using impact loading (e.g., wrecking ball) and explosive loading. This guideline will discuss the procedure step-by-step to leverage BDA with Blender and Bullet for the demolition modeling and visualization of various scenarios. The proposed approach using the computer simulation will help optimize the demolition planning using the explosives with the stress contours obtained from the simulations. The explosion is simulated based on the theory of blast waves, and a customized MATLAB code will provide further analysis for the optimization.

The guideline provides how to use the software package with examples to enhance the predictive capabilities. Hence, it makes it easier for decision-makers to get better prepared for the worst-case scenarios and choose the best demolition planning ahead depending on the target structure’s configuration and condition as well as the environmental considerations, time limits, budget, by proactively performing the visualized demolition scenarios.

1.3. Intended users

This document and software package will be of interest to highway officials, bridge (de)constructors, and structural engineers, as well as other decision-makers concerned with the safe and cost-effective bridge demolition. A final product of this project is a developed add-on to the existing software, Blender, a powerful 3D graphics software, and Bullet, a physics library. Therefore, this approach will enable to leverage the existing visualization and simulation power to model the bridge demolition with high fidelity. This guideline also provides the overview, algorithms and key functions of Blender and Bullet for the users not familiar with the software, and also discuss how BDA interacts with the existing software. Demolition using explosives, or a wrecking ball is conventionally adopted in the engineering practice, for which therefore the simulations in this guideline will be demonstrated. Several concerns related to the demolition will be addressed. For example, in the demolition using explosives, a major concern is related to using the minimum possible explosive energy that is enough to destroy the structure given the budget for the enhanced efficiency and to minimally impact the environment with less debris missiles, which will in turn protect the neighboring infrastructure as well as workers’ safety from the demolition induced hazards. Therefore, this guideline will be useful to the intended users.
2. MODELING, SIMULATION AND VISUALIZATION

The modeling, simulation, and visualization are three major steps of using the software package and herein referred to as preprocessing, analysis, and postprocessing. Modeling the geometry of the structure, assigning mechanical modeling properties, discretization based on Voronoi algorithm and creating constraints among fractured rigid bodies are called preprocessing stages and are performed in Blender. The explosion implementation and contact force retrieval cannot be performed by the existing code which are the main features introduced through Bridge Demolition Add-on (BDA) into Blender. The force by explosive blast wave is formulated based on explosion theories that are coded in Python and added to Blender. Moreover, the contact forces retrieval is not available in existing Blender and is also coded in Python as an add-on feature. BDA provides the user interface dialog box that enables user to enter modeling input parameters for explosion demolition simulation, and the post-processing panel for contact force, predicted explosion loading, estimated debris generation and force/stress contour visualization as the outputs.

2.1. Preprocessing

There are some required steps that should be done in Blender for modeling before using BDA, which referred to as preprocessing. This stage includes geometry modeling, discretization, building constraints, sequence modeling, and finally set up rendering options.

2.1.1. Geometry modeling

The geometry modeling in this project is performed by leveraging Blender, which is a free and open-source 3D computer graphics software toolset. Blender is widely used for creating animated films, interactive 3D applications and even video games. Moreover, Blender integrates Bullet as one of the main physics engines and shares some common modeling procedures, e.g., Voronoi tessellation for discretization. In this section, some basic modeling functions of Blender are introduced. Interested readers are referred to Blender User Manual [1] for further reading.

The first step models the major structural elements such as piers, pier cap and deck. A homogenized element may be modeled with equivalent mass and stiffness as far as a monolithic behavior is assumed. After all structural parts are separately modeled, they are then merged with rigid connections. Figure 1 shows a 2-span bridge with span length of 13 meters and total mass of 196 tons in Blender.
The primitive shapes such as cube, sphere, cylinder and other polyhedrons are available for modeling in Blender, meaning each shape is composed of polygonal faces, edges and vertices. Sphere is not a perfectly smooth curve, but piecewise linear. These shapes can be altered with Mesh Modeling option whereby users can move the vertices and extrude faces, etc. to change the shape and dimensions of desired object [1].
Mesh is a principal model object in Blender. Each object has an origin point which defines its location in 3D space [1]. Location, dimensions, scaling factors and other transform properties can be set in Transform tab.

![Transform tab](image)

Figure 3. Transform tab

Selecting an object can be done in two modes: Object mode and Edit mode. Each mode enables different tools and options to edit any object:

In the Object mode, adding a mesh to the world is done by hotkey Shift-A. Any type of object such as Mesh, Curve, Surface, Metaball (e.g. capsule, ellipsoid, etc.), text and others can be added to the world. Adding a force field or a lamp or a camera is also possible. Duplication of an object and regeneration of identical copies of an object can be performed by hotkey Shift-D. Hotkeys Shift-R and Shift-S can be used to rotate and scale an object respectively. Grabbing and moving object along a desired axis can be done with hotkey Shift-G-X (or Y, Z depending on the axis). Merging option can be used to create a compound object. By selecting all objects to be merged, the merging procedure can be completed by hotkey Ctrl-J [1].

There is an important consideration needs to be made when merging objects. In any merging action, it will be likely that some vertices or faces may overlap or intersect (Figure 4), which may cause technical issues when using along with the Blender fracture modifier [5] for use in model
discretization. It is recommended to use Boolean modifier (Figure 5). The Boolean modifier can apply operations such as intersection, union and difference. By using these set of operations, extra faces and vertices are simply extracted from the merged compound object (Figure 6).

Figure 4. Intersecting faces [1]

Figure 5. Boolean modifier to extract overlapping vertices, edges and faces in a compound object
Figure 6. Difference, Union and Intersect in Boolean Modifier to create a single compound shape [1] Deleting an object can be easily done by hotkey X or simply hitting Delete key. It is required to switch to Edit mode (Figure 7) to make a detailed change of the geometry by directly selecting and applying the changes to vertices, edges and faces in the object (Figure 8, 9).

Figure 7. Edit mode select mode buttons [1]

Figure 8. A sphere in Edit mode with all of vertices selected. Edges and faces can be selected separately
Any geometric operation implemented in Blender can be applied to vertices, edges and faces such as extruding, intruding, scaling, rotating, mirroring to create modified geometries (Figure 10).
Modifiers in Blender are the tools which can be utilized to alter the way objects are displayed or rendered (Figure 11). There are four categories of modifiers: *Modify, Generate, Deform* and *Simulate*:

*Modify* modifiers just affect the data of an object and do not directly influence the shape of an object.

*Generate* modifiers change the appearance of an object and add new geometry.

*Deform* modifiers change the shape of an object but do not add new geometry.

*Simulate* modifiers can add visual effects and simulations when a particle system or a physics system is defined and enabled. It can create an animated realistic ocean or generate simulation and visual effects like collision, exploding or smoke.

![Modifiers menu](image1)

Figure 11. Modifiers menu [1]

Any object modeled in Blender as a rigid body can be active or passive. An active object can be a dynamic rigid body or animated (user-controlled) rigid body. A passive rigid body is initially at rest who does not move until it reacts to a collision. Parameters related to dynamics of a rigid body can be set in Rigid Body Modifier tab as shown in Figure 12.
Physics modeling parameters such as mass, friction coefficient, and bounciness (coefficient of restitution or contact damping) can be set. There are options called Enable Deactivation and Start Deactivated that play an important role in the demolition simulations. If these options are not checked, the pre-fractured bridge may start to collapse from the very beginning of simulation, even before the wrecking ball has not touched the structure. This behavior also strongly depends on the constraints defined for the shards. Tuning of these options will be further discussed in the later sections.
2.1.2. Discretization and fracturing

There are two methods available in Blender to implement the discretization in the developed bridge model: (a) Cell Fracture add-on and (b) Fracture Modifier which is a more recent development compared to the former. We will briefly introduce the (a) Cell Fracture add-on first as below.

The Cell Fracture add-on is not a default option in Blender, which therefore should be imported first. The import procedure is shown in Figure 14, which can be done by checking Object: Cell Fracture in the Add-ons tab as below. Then Cell Fracture add-on is added to the Tools bar as shown in Figure 15.

Figure 13. Collapse of the bridge model due to pre-fracturing and not selecting start deactivated option

Figure 14. Enabling cell fracture add-on in Blender’s user preference.
Figure 15. Cell Fracture add-on enabled in the Tools bar.

Figure 16. Cell Fracture features and parameters.

While bridge object being selected, clicking the Cell Fracture opens the Cell Fracture window as shown in Figure 16. The Source Limit parameter in the add-on determines the maximum number of shards that can be generated. Noise value is a value between 0 and 1, which accounts for irregularity of fracture sizes. The value 0 indicates completely uniform shards in terms of shape and size, while 1 indicates the maximum nonuniformity. The Recursion parameter helps to generate smaller shards. The shards’ mass can be set uniform or proportional to their volume by controlling
the Mass parameter. By checking the Recenter option, the physics properties of each shard are calculated based on its own origin. Checking the Next layer option makes the discretization to be shown in the second layer rather than the main layer of the scene. By pressing OK, the bridge is discretized. An example is shown in Figure 17.

![Image of discretized bridge](image)

**Figure 17.** Discretized bridge by using cell fracture add-on

Now, rigid body properties will be assigned to new rigid bodies generated by fracturing, which can be done by using Physics Body Tools bar shown in Figure 18. Selected the shards can be set as active objects with the Add Active option. The volume proportional mass can be calculated with Calculate Mass option and then Concrete option can be selected as the designated material.

![Image of Physics Body Tools](image)

**Figure 18.** Physic Body Tools bar for assigning rigid body properties
In the recently developed (b) Fracture Modifier, a set of fracture algorithms are available. In this study, Voronoi Boolean algorithm is selected for the model pre-fracturing. There are also some options available for solver from which Carve is selected. The number of shards can be chosen to determine how the bridge model is to be fractured, which can be selected depending on the computational cost and simulation fidelity. A large number of shards will be more realistic, but is computationally much expensive, as it requires to simulate more number of broken pieces and their interactions in every time step.

![Figure 19. Discretization of a 2-span bridge model by Fracture modifier](image)

The constraints between the shards can be set in the Fracture Constraint Settings tab (Figure 21). Here, one can enable to use constraints, set them breakable, and select the type of the constraints between the shards. The example in Figure 21 shows Hinge type is selected as the Constraint Type and Vertex is chosen as the Constraint Method. Number of constraints per mesh island can be also set. The threshold where the constraints to be broken can be set in Constraint Breaking Settings tab (Figure 22).
2.1.3. Building constraints

With the pre-fracturing performed, the bridge model is geometrically discretized into hundreds of separate rigid bodies, then the constraints between these objects need to be set up. To this end, Bullet Viewport Constraints Tool add-on [6] is used. The add-on is not a default option in Blender, which therefore should be imported first as well (Figure 23). Once installed, the option is added on the left tool bar in Blender’s 3D view as shown in Figure 24.
The corresponding shards need to be selected to assign the building constraints in the discretized bodies. The same constraints may be applied to all the shards in the bridge model, or different constraints may be selectively assigned to different parts of the bridge. The Neighbor Limit value in the Bullet Constraint Tools window (Figure 24) determines the maximum number of surrounding shards to be considered for building a constraint around the target shard, i.e., for nonlocal interaction of rigid bodies. By checking Friction, a frictional contact is considered between the shards, for which the friction coefficient (such as Coulomb friction) and the restitution values need to be set. The Enable Deactivation option enables the discretized model is dynamically deactivated if the external forces applied at the interface is not high enough compared to the constraint threshold, i.e., not strong enough to break the constraint. This option helps keep the computational cost more manageable by lowering the number of the necessary contact detection that is computationally expensive in general. Linear Velocity and Angular Velocity are the constraint thresholds for activation to translation and rotation respectively. Constraint Type option enables to choose desired type of constraints including fixed, hinge, slider, point, etc. Pressing X Constraints button at the top shows the built constraints on the top right corner of Blender graphical user interface. An example is shown in Figure 25.
2.1.4. Sequence modeling

There are numerous cases that movement of an object needs to be manually animated in a simulation. For example, a wrecking ball is pulled up by a cable connected to a crane, and then dropped by gravity to impact the bridge structure. Then it’s pulled up again for the next hit. This sequentially controlled movement can be modeled in Blender. The object for animation should be selected and checked as Animated in Dynamics tab (Figure 26). By using Dope Sheet tab and setting Keyframes in a series of desired frames, one can change the dynamics of the wrecking ball from animated to dynamic motion or vice versa (Figure 27).
2.1.5. Rendering

Once the simulation of bridge demolition is completed, the results needs to be visualized for which sophisticated rendering is needed. One can monitor how the rendered image of an object appears by controlling the four factors: (i) camera, (ii) lighting of the scene, (iii) object’s material and (iv) general render settings. Blender adopts two different render engines: Blender Renderer and Cycles. Regardless what render engine is used, the rendering commonly is performed in the following steps: First, the location of the (i) camera is defined. The (ii) lighting of the scene is then determined. Instead of using conventional lighting options, emission planes can be used to provide enough light to the scene. This is done by adding some number of plane meshes in the scene with relatively greater dimensions compared to the developed bridge model, and selecting their positions and orientations in a way that they can emit sufficient amount of light to the scene (Figure 28). Then one should go to Material modifier and define a new material and then select Emission and set the strength of the emission in the Surface tab.

The (iii) materials then can be assigned to the objects. It’s also possible to assign a texture, which is optional. Material in Blender can be defined by clicking the Materials button (Figure 29). To
add a new material, click + in the material box. A material has a wide range of properties such as preview, diffuse shaders, color ramps, shading, transparency, etc. which can affect the final rendering.

![Materials Panel](image)

**Figure 29. Materials Panel**

User may switch to Cycles Render mode and set the (iv) general render settings like start and end frame, frame rate, processor device, address to save the video, etc. in Render settings tab (Figure 30).

![Rendering settings](image)

**Figure 30. Rendering settings**

2.2. **Bridge Demolition Add-on**

2.2.1. **Overview**

Bridge Demolition Add-on (BDA) version 1.0.0 has been developed from this project. It was designed as an add-on to provide enhancement to Blender 2.79, whereby the computational framework is customized for numerical modeling and simulation of various bridge demolition scenarios.
BDA helps model two popular demolition methods for bridge demolition: (i) explosive demolition and (ii) mechanical demolition with a wrecking ball. BDA is also equipped with additional post-processing features such as contact force computation in the structure during demolition. The Blender’s physics engine Bullet works on the first order dynamics where the collision impulse is the primary variable, where the contact force computation is bypassed during the simulation. However, the force is the key engineering information for demolition planning, e.g., to determine the weight of wrecking ball to be used, amount of explosives, etc., and therefore is of great importance from the structural engineering aspects. This information can be visualized using the force contour with BDA. Another significant feature is Debris Propagation Log, which enables the decision-makers, engineers and contractors to know how far the demolition induced debris travels. BDA user interface and features are further discussed in following sections.

2.2.2. BDA installation

BDA can be installed on Blender as follows:

Open Blender, go to the File tab, select User Preferences (Figure 31). Then select Install Add-on from File in the Add-ons section as below.

![Blender user preferences, Add-ons](image)

Figure 31. Blender user preferences, Add-ons

Then select the path to the zip file of BDA downloaded on your computer as shown in Figure 32.
Select the add-on file and install it. Then you can see the add-on name and details as in Figure 33.

Figure 33. Bridge Demolition Add-on’s name and description are shown for installation
Save User Settings, and close Blender User Preferences Window. Then Bridge Demolition Add-on is shown on the Blender’s tool bar as BDA (Figure 34).

![Figure 34. BDA added to Blender’s UI](image)

2.2.3. **User interface and features**

The user interface (UI) created for Bridge Demolition Add-on is shown in Figure 35.
As shown in Figure 35, the BDA UI has three major panels: Explosion, Post Processing and Frame Change Event Handler.

In Explosion panel, there is a check box named Explosion. If the user is going to perform a simulation of demolition by explosives, this box should be checked. It is worth to note accidentally checking this box causes no problem even if the demolition method is not by explosion. Instead, it will just slow down the computation process because of some initializations required for explosion simulation. User can control the simulation of explosives with two parameters. The first parameter is the location of explosives (or explosion source points). With Add Source Point clicked in the UI, the user can define unlimited numbers of explosion source points. Figure 35 shows a new row is generated with the Add Source Point activated, and the user can define explosion source point location in Cartesian coordinate (with X, Y and Z coordinates) and the amounts of explosives related to the entered source points. The amount is to be entered as the TNT mass of explosives in kg.
After defining desired explosion source points, Build Force Objects should be clicked to create a set of empty objects to be used for applying explosive force in 3D. This will be further elaborated in following sections. User can click Show Mass of Particles to calculate the total mass of bridge model, which is optional.

Post Processing panel introduces two additional features. The panel is mainly to compute the contact forces among fractured bodies. Contours check box enables the contact force visualization via contours during the simulation. The Set Force Log Filepath option is to write the computed contact forces in each time step of simulation on a file whose path is selected by the user. The Set Debris Log Filepath option will record the locations of rigid bodies in the first and last time steps on a file whose path is selected by the user. This will allow to calculate maximum distance traveled by a fractured part and enable to predict debris impact on the surroundings due to the demolition. This information will be vital in the demolition project because stakeholders may want to make sure the surrounding infrastructure is not affected by demolition-induced debris.

The Frame Change Event Handler panel is designated for registering and unregistering event frame handlers. Event frame handlers play important role to transfer data and calculated values from one timestep to another to be used for the contour development.
2.2.4. Contact force calculation and contour visualization

One of the main features of BDA is its capability of computing the contact forces between fractured bodies during the simulation and visualizing the information with the contours in each time step. Figure 36 shows an example of the contact force contour created by BDA due to the wrecking ball generated impact on the bridge deck of a single span.

![Contact force contour created by BDA; The bridge is under the wrecking ball generated impact](image)

Bullet Physics as the physics engine integrated into Blender, which is an impulse-based simulator that bypasses the contact force computation. However, the contact force is a critical information needed for the engineering applications, for which BDA has been developed to retrieve the lost contact force information. The theory is based on Lee and Hashash [7], and interested readers are directed to the reference for the further technical details. The code for contact force calculation and contour visualization has been implemented in Python using Blender’s API and is presented here. Some parts of the contour visualization are loosely based on [8].
2.2.5. **Implementation of capability to simulate the demolition by explosion**

Explosion is one of the major methods of bridge demolition and one of the major features implemented in BDA. Figure 37 shows an example of simulation of demolition by explosives with BDA.

![Figure 37](image1.png)

Figure 37. An explosion simulation of a single span bridge performed with support of BDA

![Figure 38](image2.png)

Figure 38. Contact force contour visualization by the explosives
Explosion is a sudden energy release which generates a blast wave. The blast wave is assumed to show a spherical propagation with a pressure profile. Below is some key formula of the blast wave theory by explosion. Interested readers are referred to [9] for more reading.

The pressure profile generated by explosion is defined by Friedlander equation:

\[ p(t) = p^0(1 - \frac{t}{t_d})e^{-\alpha t/t_d} \]  

where \( p \) is the instantaneous overpressure at time \( t \); \( p^0 \) is the maximum or peak overpressure observed when \( t \) is zero at the moment of explosion; \( t_d \) is the time duration; and \( \alpha \) is a constant [9].

The corresponding force on the surrounding bodies can be computed from the pressure with the following equation:

\[ \vec{F} = p(t) A \frac{\vec{R}}{||\vec{R}||} \]  

where \( p(t) \) is the pressure generated by explosion; \( A \) is the projected area of a body; and \( \vec{R} \) is the position vector that indicates the position of the body from source of explosion.

The overpressure vs. distance relation for an explosion can be shown by the following equation:

\[ \frac{p^0}{P_a} = \frac{808 \times [1 + \left( \frac{Z}{4.5} \right)^2]}{\sqrt{1 + \left( \frac{Z}{0.048} \right)^2} \times \sqrt{1 + \left( \frac{Z}{0.32} \right)^2} \times \sqrt{1 + \left( \frac{Z}{1.35} \right)^2}} \]  

where \( \frac{p^0}{P_a} \) is the ratio of peak overpressure to ambient atmospheric pressure; and \( Z \) is called scaled-distance that is the actual distance in meters [9].

The duration, \( t_d \), is the time duration in which explosive forces are applied and hence is an index of how damaging the explosion can be. It’s common to assume the positive overpressure phase as the time duration of the explosion:
\[
t_d = \frac{980 \times \left( 1 + \left( \frac{Z}{0.54} \right)^{10} \right)}{W^{\frac{1}{3}} \times \left[ 1 + (\frac{Z}{0.02})^3 \right] \times \left[ 1 + (\frac{Z}{0.74})^6 \right] \times \sqrt{1 + (\frac{Z}{6.9})^2}}
\]

where \( \frac{t_d}{W^{\frac{1}{3}}} \) is the explosion duration time in milliseconds for one kilogram TNT explosion and \( Z \) is the scaled distance in meters; and \( t_d \) is the time duration of explosive forces.

The scaling law for explosions yields as below based on conservation of momentum and geometric similarity:

\[
D_s = \frac{f_d \times D_a}{W^{\frac{1}{3}}}
\]

where \( D_s \) is the scaled distance to the source of explosion; \( D_a \) is the actual distance to the source of explosion; and \( f_d \) is the transmission factor which is assumed to be equal to 1, which means that the change of atmospheric density due to explosion is ignored.

Figure 39 shows how overpressure ratio varies with time for different scaled distances generated by a \( W \) equal to 10,000 kg of TNT. Figure 40 shows how overpressure ratio due to explosion varies by scaled distance for different explosion intensities. Explosion intensity is in terms of \( W \) in kilograms of TNT. These examples of contour are created by the MATLAB code implemented as a part of this project. With use of these codes, users can estimate the expected overpressure ratio in a specific distance from explosion source point with respect to explosion intensity applied. Figure 41 shows an example regarding how much explosion intensity is required given a target overpressure (i.e. a desired amount of damage) to be applied in a specific distance.

As a real example, the demolition of a two-span bridge (each span has a length of 20 meters) is considered. Total amount of 1000 kg of explosives is installed at both ends and mid piers of the bridge. Figure 42 shows how overpressure ratio varies by scaled distance due to explosion at each pier and Figure 43 shows the total overpressure ratio along the bridge due to the explosion.
Figure 39. Variation of overpressure ratio vs time for different scaled distances

Figure 40. Explosion intensity contours
Figure 41. Overpressure ratio contours

Figure 42. Variation of overpressure vs. scaled distance for a 2-span bridge exploded by 1 ton of explosives
One of the main challenges in developing code for implementing the explosion is the calculation of projected areas of each fractured shard with regards to explosion induced pressure. The convex hull is computed from the discretized shards’ geometry, which is then used to estimate the explosion force applied to the body. BDA is equipped with the functions that can precisely calculate the projected area of convex hull and update the information each time step with respect to the updated locations and orientations of the fractured bodies. Projected area of a convex hull with respect to a designated explosion source point is calculated as in Equation:

$$PA = \frac{1}{2} \sum_{i=1}^{k} (|\hat{n}_i \cdot \hat{n}| S_i)$$

where $PA$ is the projected area of the convex hull; $k$ is the number of faces of a convex hull; $\hat{n}$ is normal vector of the projection plane (which is simply the subtraction of fracture center of mass location vector and explosion source point location vector); and $\hat{n}_i$ is the normal vector of the face; and $S_i$ is the face area.
Projected area computation is coded in Python using Blender’s API as follows:

```python
# bmesh from object

def bmesh_from_object(ob):
    matrix = ob.matrix_world
    me = ob.to_mesh(bpy.data.scenes['Scene'], apply_modifiers=True, settings='PREVIEW')
    me.transform(matrix)
    bm = bmesh.new()
    bm.from_mesh(me)
    bpy.data.meshes.remove(me)
    return bm

# dot product

def dot_product(vec1, vec2):
    return sum

# projected area

def projected_area(obj, projection_normal_vector):
    bm = bmesh_from_object(obj)
    bm.normal_update()
    print('bmesh:', bm)

    pa = 0
    for f in bm.faces:
        face_area = f.calc_area()
        face_normal = f.normal

        # compute projected area
        face_pa = abs(dot_product(projection_normal_vector, face_normal)) * face_area
        pa += face_pa
        print('face index:', f.index, ' and face area:', face_area, 'face normal:', face_normal, 'face pa:', face_pa)

    print
    bm.free()

    pa = pa/2
    return pa
```

---

39
Explosion force applied to any fracture from a source point is calculated by the function implemented as follows:

```python
def explosion(m_origin, m_source, time_elapsed, projected_area, amount):
    #btVector3
    w = amount;
    Distance[0] = m_origin[0] - m_source[0]
    Distance[1] = m_origin[1] - m_source[1]
    distance = pow((Distance[0] * Distance[0] + Distance[1] * Distance[1] + Distance[2] * Distance[2]), 0.5);
    distance /= pow(w,1/3);
    area = projected_area;
    ambientPressure = 101325.0;
    peakOverpressure = ambientPressure * 808 * ((pow(distance / 4.5, 2) + 1) / pow((1 + (pow(distance / 0.048, 2))), 0.5) * pow((1 + (pow(distance / 0.32, 2))), 0.5)) * pow((1 + (pow(distance / 1.35, 2))), 0.5));
    millisecond_to_second = 0.001
    td = millisecond_to_second * 980 * (1 + pow(distance / 0.54, 10)) / ((1 + pow(distance / 0.02, 3.0)) * (1 + pow(distance / 0.74, 6)) * (1 + pow(distance / 6.9, 2), 0.5)) * pow(w, 1/3);
    beta = -1.0;
    #Friedlander Equation
    pressure = ambientPressure + peakOverpressure * (1 - time_elapsed / td) * math.exp(beta * time_elapsed / td);
    explosionforcex = pressure * area * Distance[0] / pow((Distance[0] * Distance[0] + Distance[1] * Distance[1] + Distance[2] * Distance[2]), 0.5);
    explosionforcey = pressure * area * Distance[1] / pow((Distance[0] * Distance[0] + Distance[1] * Distance[1] + Distance[2] * Distance[2]), 0.5);
    explosionforcez = pressure * area * Distance[2] / pow((Distance[0] * Distance[0] + Distance[1] * Distance[1] + Distance[2] * Distance[2]), 0.5);
    m_explosionForce = [explosionforcex, explosionforcey, explosionforcez];
    return m_explosionForce;
```

Code development of explosion implementation in Blender’s API is as follows, where the Force field in Blender has been leveraged:

```python
# Field Force Properties
FIELD_HIDE = True
FIELD_TYPE = 'FORCE'
FIELD_Z_DIRECTION = 'BOTH'
```
5. FIELD_SHAPE = 'POINT' # 'PLANE'
6. FIELD_STRENGTH = 0.0
7. FIELD_FLOW = 0.0
8. FIELD_SEED = 40
9. FIELD_APPLY_TO_LOCATION = True
10. FIELD_APPLY_TO_ROTATION = True
11. FIELD_FALLOFF_TYPE = 'TUBE'
12. FIELD_FALLOFF_POWER = 10.0
13. FIELD_USE_MIN_DISTANCE = True
14. FIELD_DISTANCE_MAX = 5 # 0.2
15. FIELD_USE_RADIAL_MIN = False
16. FIELD_USE_RADIAL_MAX = True
17. FIELD_RADIAL_MAX = 2 # 0.2
18. FIELD_RADIAL_FALLOFF = 0.0 # real gravitational falloff = 2!!
19. 
20. 
21. # Rotation
22. ROTATION_90 = 1.5707961320877075
23.  
24. class ExplosionOperator (bpy.types.Operator):
25.     bl_idname = "wm.explosion_operator"
26.     bl_label = "Add Source Point"
27. 
28.     def execute(self, context):
29.         scene = context.scene
30.         explosion_tool = scene.explosion_tool
31.         i = 0
32.         for item in context.scene.collection:
33.             print(item.source[0], item.source[1], item.source[2], context.scene.amountcollection[i].amount)
34.             i = i + 1
35.         return {'FINISHED'}
36. 
37. 
38. 
39. 
40. def build_all_force_objects():
41.     for obj in bpy.data.objects:
42.         if obj.type != 'MESH':
43.             continue
44.         if not hasattr(obj.rigid_body, 'type'):
45.             continue
46.         if obj.rigid_body.type != 'ACTIVE':
47.             continue
48.         
49.         # create_empty_obj
50.         create_empty_obj(obj, 0)
51.         create_empty_obj(obj, 1)
52.         create_empty_obj(obj, 2)
53. 
54. 
55. 
56. ################################################################################
57.  
58. # create empty object
59. 
60. def create_empty_obj(parent_obj, axes_index):
61.     emptyname = 'Empty_' + str(axes_index) + '_' + parent_obj.name
62.     emptyname = bpy.data.objects.get(emptyname) is None:
63.         bpy.ops.object.empty_add(type='PLAIN_AXES')
emptyobj = bpy.data.objects[bpy.context.active_object.name]
emptyobj.name = emptyname
emptyobj.select = False
emptyobj.hide = FIELD_HIDE
#emptyobj.parent = parent_obj
emptyobj.location = parent_obj.location
emptyobj.rotation_euler[0] = 0
emptyobj.rotation_euler[1] = 0
emptyobj.rotation_euler[2] = 0
if (axes_index < 2):
    emptyobj.rotation_euler[axes_index] = ROTATION_90
emptyobj.field.type = FIELD_TYPE
emptyobj.field.z_direction = FIELD_Z_DIRECTION
emptyobj.field.shape = FIELD_SHAPE
emptyobj.field.flow = FIELD_FLOW
emptyobj.field.seed = FIELD_SEED
emptyobj.field.apply_to_location = FIELD_APPLY_TO_LOCATION
emptyobj.field.apply_to_rotation = FIELD_APPLY_TO_ROTATION
emptyobj.field.falloff_type = FIELD_FALLOFF_TYPE
emptyobj.field.falloff_power = FIELD_FALLOFF_POWER
emptyobj.field.use_min_distance = FIELD_USE_MIN_DISTANCE
emptyobj.field.use_max_distance = FIELD_USE_MAX_DISTANCE
emptyobj.field.distance_max = FIELD_DISTANCE_MAX
emptyobj.field.use_radial_min = FIELD_USE_RADIAL_MIN
emptyobj.field.use_radial_max = FIELD_USE_RADIAL_MAX
emptyobj.field.radial_max = FIELD_RADIAL_MAX
emptyobj.field.radial_falloff = FIELD_RADIAL_FALLOFF
emptyobj.field.strength = explosion_force[str(i)]
if i < 2:
    emptyobj.field.strength = explosion_force[abs(i-1)]
else:
    emptyobj.field.strength = explosion_force[i]

# explosion

def explosion(m_origin, m_source, time_elapsed, projected_area, amount):
    Distance = [0,0,0] #btVector3
    w = amount;
    Distance[0] = m_origin[0] - m_source[0]
    Distance[1] = m_origin[1] - m_source[1]
    distance = pow((Distance[0] * Distance[0] + Distance[1] * Distance[1] + Distance[2] * Distance[2]), 0.5);
distance /= pow(w, 1/3);

area = projected_area;

ambientPressure = 101325.0;

peakOverpressure = ambientPressure * 808 * (pow(distance / 4.5, 2) + 1) / (pow((1 + (pow(distance / 0.048, 2)), 0.5) * pow((1 + (pow(distance / 0.32, 2)), 0.5) * pow((1 + (pow(distance / 1.35, 2)), 0.5)));

millisecond_to_second = 0.001

td = millisecond_to_second * 980 * (1 + pow(distance / 0.54, 10)) / ((1 + pow(distance / 0.02, 3.0)) * (1 + pow(distance / 0.74, 6)) * (pow(1 + pow(distance / 6.9, 2), 0.5)) * pow(w, 1/3));

beta = -1.0;

# Friedlander Equation
pressure = ambientPressure + peakOverpressure * (1 - time_elapsed / td) * math.exp(beta * time_elapsed / td);

explosionforcex = pressure * area * Distance[0] / pow((Distance[0] * Distance[0]) + Distance[1] * Distance[1] + Distance[2] * Distance[2], 0.5);
explosionforcey = pressure * area * Distance[1] / pow((Distance[0] * Distance[0]) + Distance[1] * Distance[1] + Distance[2] * Distance[2], 0.5);
explosionforcez = pressure * area * Distance[2] / pow((Distance[0] * Distance[0]) + Distance[1] * Distance[1] + Distance[2] * Distance[2], 0.5);

m_explosionForce = [explosionforcex, explosionforcey, explosionforcez];

return m_explosionForce;

# bmesh from object

def bmesh_from_object(ob):
    matrix = ob.matrix_world
    me = ob.to_mesh(bpy.data.scenes['Scene'], apply_modifiers=True, setings='PREVIEW')
    me.transform(matrix)
    bm = bmesh.new()
    bm.from_mesh(me)
    bpy.data.meshes.remove(me)
    return bm

# dot product

def dot_product(vec1, vec2):
    return sum

# projected area
def projected_area(obj, projection_normal_vector):
    bm = bmesh_from_object(obj)
    bm.normal_update()
    print("bmesh:", bm)
    pa = 0
    for f in bm.faces:
        face_area = f.calc_area()
        face_normal = f.normal
        # compute projected area
        face_pa = abs(dot_product(projection_normal_vector, face_normal)) * face_area
        pa += face_pa
        print('face index:', f.index, 'and face area:', face_area, 'face normal:', face_normal, 'face pa:', face_pa)
    print
    bm.free()
    pa = pa/2
    return pa

# do_explosion(current_frame):
print("do:", current_frame)
# deselect all of the objects
bpy.ops.object.select_all(action='DESELECT')
time_scale = bpy.context.scene.rigidbody_world.time_scale
steps_per_second = bpy.context.scene.rigidbody_world.steps_per_second
time_elapsed = current_frame * time_scale / steps_per_second
if (not bpy.context.scene.explosion_tool.logfilepath):
    bpy.context.scene.explosion_tool.logfilepath = "forcelogfile_explosion.log"
    f = open(bpy.context.scene.explosion_tool.logfilepath+"_explosion.log", "a")
    f.write("name, projected area, explosion force, projection_normal_vector\n")
    f.write("current_frame: ' + str(current_frame) + ' elapsed time: ' + str(time_elapsed) + '\n")
    for obj in bpy.data.objects:
        if obj.type != 'MESH':
            continue
        if not hasattr(obj.rigid_body, 'type'):
            continue
        if obj.rigid_body.type != 'ACTIVE':
            continue
        projection_normal_vector = obj.rigid_body.location - SOURCE_POINT
pa = projected_area(obj, projection_normal_vector)

index = 0

total_explosion_force = [0, 0, 0]

for item in bpy.context.scene.collection:
    amount = bpy.context.scene.amountcollection[index].amount
    explosion_force = explosion(obj.rigid_body.location, item.source, time_elapsed, pa, amount)
    print("explosion_force:", explosion_force)
    for i in range(3):
        total_explosion_force[i] = total_explosion_force[i] + explosion_force[i]
    print("total_explosion_force:", total_explosion_force)
    index = index + 1

print('name:', obj.name, 'projected area:', pa, 'explosion force:', total_explosion_force, 'projection_normal_vector:', projection_normal_vector)
f.write(obj.name + ',' + str(pa) + ',' + str(explosion_force) + ',' + str(projection_normal_vector) + '\n')

# apply force
update_empty_objects(obj, total_explosion_force)

Figure 44. Leaving Explosion unchecked in Explosion panel for wrecking ball demolition simulation

2.3. How to use Bridge Demolition Add-On

After finishing preprocessing and getting the add-on installed and added to Blender (Section 2.2.2), the bridge model is prepared for a demolition simulation by BDA. If the desired method of demolition is mechanical method using a wrecking ball, leave the Explosion option unchecked in Explosion panel as discussed above and shown in Figure 44.
User may click Show Total Mass of Particles button as shown in Figure 45 to compute the exact mass of the discretized bridge model.

![Figure 45. Computed mass of the developed bridge model](image)

Contact force contour visualization is checked by default, which however may be unchecked to disable the contour visualization during simulation. Set the file path by pressing Set Debris Log Filepath to get the debris propagation data. This will write the locations of rigid bodies in the first and the last frame to a text file whose path is selected by the user. This will allow for estimating the maximum distance traveled by each fractured body. Set the file path by pressing Set Force Log Filepath to get the contact forces calculated during the simulation for each frame. This will write the calculated contact forces each timestep of the simulation to a file whose path is selected by the user. Figure 46 shows an example where the paths are set for the output files in the user interface.

![Figure 46. Paths set for the output files in the interface](image)
The model’s constraints can be hidden by opening the Python console in Blender and enter command lines as shown in Figure 47.

![Figure 47. Hiding constraints](image)

Event handlers can be automatically created by the program by activating Register Handler in the last panel. Then play the simulation.

At the beginning of simulation, the contour will be shown all in dark blue (Figure 48) showing the minimum contact force values. Figure 49 shows contact force contours as a wrecking ball hits the bridge deck. The maximum contact force values are indicated in darker red.

![Figure 48. Minimum contact force values](image)
For the demolition by explosion, the Explosion option should be checked in the panel. With the Add Source Point option, (unlimited number of) new explosion source points can be defined. As shown in Figure 50, by clicking Add Source Point, a new row is generated and the user can define an explosion source point location by entering X, Y and Z coordinates and the amounts of explosives related to each source point. After defining the desired explosion source points, Build Force Objects can be clicked. This will create empty objects which are used for applying the explosion force in three dimensions.
Paths and filenames need to be set for debris propagation data and computed contact force as simulation output using Set Debris Log Filepath and Set Force Log Filepath (Figure 51). With Explosion option checked, an extra text file will be created in the same path which contains the magnitude of explosive force applied, the projected area and the normal vector of each rigid body in each frame from beginning till the end of simulation. With Register Handler activated in the last panel, event handlers are automatically created by the program. Figure 52 shows a simulation example of demolition by explosives.
Figure 52. Explosion simulation example performed by BDA

2.4. Calibration, verification and examples

The fidelity of modeling parameters affects the simulation’s fidelity. These parameters should be calibrated and fine-tuned by reproducing the actual demolition data or video recordings. Some of these parameters are Blender’s internal parameters that need to be found through some trial and error process because some of them are numerical artifacts. Table 1 lists Blender’s internal parameters as well as the physical parameters. More details of Blender’s internal parameters can be found in [1].

Table 1. List of simulation parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrecking Ball Mass</td>
<td>External</td>
</tr>
<tr>
<td>Bridge Total Mass</td>
<td>External</td>
</tr>
<tr>
<td>Bridge Model Geometry</td>
<td>External</td>
</tr>
<tr>
<td>Amounts of Explosives</td>
<td>External</td>
</tr>
<tr>
<td>Explosion Source Points</td>
<td>External</td>
</tr>
<tr>
<td>Source Limit (Number of fractures)</td>
<td>Cell Fracture</td>
</tr>
<tr>
<td>Noise</td>
<td>Cell Fracture</td>
</tr>
<tr>
<td>Feature</td>
<td>Type</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Neighbor Limit</td>
<td>Bullet Constraints Tool</td>
</tr>
<tr>
<td>Search Radius</td>
<td>Bullet Constraints Tool</td>
</tr>
<tr>
<td>Friction Coefficient</td>
<td>Bullet Constraints Tool</td>
</tr>
<tr>
<td>Bounciness</td>
<td>Bullet Constraints Tool</td>
</tr>
<tr>
<td>Collision Margin</td>
<td>Bullet Constraints Tool</td>
</tr>
<tr>
<td>Activation Linear Velocity</td>
<td>Bullet Constraints Tool</td>
</tr>
<tr>
<td>Activation Angular Velocity</td>
<td>Bullet Constraints Tool</td>
</tr>
<tr>
<td>Constraints Type</td>
<td>Bullet Constraints Tool</td>
</tr>
<tr>
<td>Constraint Breaking Threshold</td>
<td>Bullet Constraints Tool</td>
</tr>
<tr>
<td>Rigid Body Type</td>
<td>Rigid Body</td>
</tr>
<tr>
<td>Collision Shape</td>
<td>Rigid Body Collisions</td>
</tr>
<tr>
<td>Field_Hide</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Type</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Z_Direction</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Shape</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Strength</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Flow</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Seed</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Apply_To_Location</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Apply_To_Rotation</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Falloff_Power</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Falloff_Type</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Use_Min_Distance</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Use_Max_Distance</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Distance_Max</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Use_Radial_Min</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Use_Radial_Max</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Radial_Max</td>
<td>Force Field</td>
</tr>
<tr>
<td>Field_Radial_Falloff</td>
<td>Force Field</td>
</tr>
<tr>
<td>Rotation_90</td>
<td>Force Field</td>
</tr>
</tbody>
</table>
A set of simulations have been carried out to reproduce the I-235 Bridge demolition project. The bridge is modeled with the actual geometry (such as span length, deck width, pier height, etc.), then material properties are then entered, and discretized in Blender. The demolition of I-235 Bridge demolition was done by utilizing a wrecking ball, for which three set of simulations are carried out for one, two, and three span bridges, respectively. Figure 53 and 54 show the simulation of the one span bridge demolition after the other decks already destructed. Figure 55 and 56 show the visualized demolition simulation. There are a set of physics modeling parameters need to be calibrated for realistic simulations. Table 2 shows the key parameter values used for the reasonable simulation fidelity. Figure 57 and 58 show the I-235 Bridge with two spans remaining with the other decks destructed. Figure 59 and 60 show the corresponding visualization performed using the exactly same set of physics modeling parameters calibrated with the demolition simulation of single span I-235 Bridge (Figure 53 and 54). The impact of the location on the three-span bridge is then simulated where the influence of different impact points of the wrecking ball is also estimated. Four different simulations are performed for four different impact locations as shown in Figure 61 to 64.

Figure 53. I-235 Bridge with a span about to be demolished by dropping a wrecking ball
Figure 54. The one last remaining deck of I-235 Bridge demolished by the wrecking ball

Figure 55. The one span I-235 Bridge modeled
Figure 56. Simulated demolition of the one last remaining deck of I-235 Bridge

Table 2. List of key parameter values for the demolition simulation of I-235 Bridge

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bridge Mass (single span)</td>
<td>107,000 kg</td>
</tr>
<tr>
<td>Wrecking Ball Mass</td>
<td>10,000 kg</td>
</tr>
<tr>
<td>Fracturing Algorithm</td>
<td>Voronoi Boolean</td>
</tr>
<tr>
<td>Shard Count</td>
<td>200</td>
</tr>
<tr>
<td>Constraint Method</td>
<td>Vertex</td>
</tr>
<tr>
<td>Constraint Type</td>
<td>Hinge</td>
</tr>
<tr>
<td>Constraint Limit per mesh island</td>
<td>50</td>
</tr>
<tr>
<td>Search Radius</td>
<td>1.00 m</td>
</tr>
<tr>
<td>Constraint Breaking Threshold</td>
<td>10000 kgf</td>
</tr>
</tbody>
</table>
Figure 57. I-235 Bridge with two spans about to be demolished by dropping a wrecking ball

Figure 58. The one of the remaining decks of I-235 Bridge demolished by the wrecking ball
Figure 59. The two span I-235 Bridge modeled

Figure 60. Simulated demolition of the one of remaining decks in I-235 Bridge
Figure 61. Simulated demolition of the three span I-235 Bridge, where the wrecking ball is dropped near the end of the bridge.

Figure 62. Simulated demolition of the three span I-235 Bridge, where the wrecking ball is dropped near the center of the neighboring bridge deck.
Figure 63. Simulated demolition of the three span I-235 Bridge, where the wrecking ball is dropped on the mid bridge deck; the hitting point is between the center of the mid bridge deck and the neighboring deck on the left in the figure.

Figure 64. Simulated demolition of the three span I-235 Bridge, where the wrecking ball is dropped at the center of mid bridge.
Figure 65 shows the dismantling demolition using a jackhammer. The corresponding simulation is shown in Figure 66 that presents a good level of realism. Reproduction of this type of demolition requires relatively fine mesh compared to the wrecking ball demolition simulation above, which is therefore computationally more expensive in general.

Figure 65. Dismantling demolition using a jackhammer

Figure 66. Simulated demolition by jackhammer
3. PHYSICS IMPLEMENTATION WITH BULLET

3.1. Bullet physics library

Bullet Physics is an open source and free library written in C++, which is widely used as physics engine on various platforms including Play Station 3, Xbox 360, PC, Linux, Mac OSX, Android and iPhone for developing games. Bullet Physics is under zlib license, and its modification can be freely redistributed as far as the terms in the public license are agreed. Interested readers are directed to [10] for further details.

3.2. Data types and math library

The basic data types in Bullet are btScalar, btVector3, btQuaternion and btMatrix3X3 and btTransform. These types along with memory management and containers can be found in Bullet/src/LinearMath.

btScalar data type is used for compiling the library in single-floating point precision and double precision. It is a typedef to float and can be also used for double precision arithmetic.

btVector3 is defined for indicating vectors and 3D positions. It includes three scalar components x, y, and z, and an extra component for considering alignment and single instruction multiple data compatibility. A variety of operations such as adding, subtracting and cross product of two vectors can be done with it.

btQuaternion and btMatrix3X3 are defined and used for showing 3D orientations and rotations.

btTransform is used for transferring vectors between different coordinates and is a container of position and orientation.

These types along with memory management and containers can be found in Bullet/src/LinearMath.

3.3. Collision detection

Collision detection is one of the most important and computationally expensive part of the physically based engineering simulation. It is generally desired to obtain collision impulse produced between the pair of polygonal rigid bodies in collision, prevent them penetrating each other. Bullet splits contact detection to two phases: Broad Phase and Narrow Phase. In simulation two rigid bodies are defined as colliding each other when the distance between their centers of mass is within a tolerance (broad phase) or their geometries are intersecting (narrow phase) [11].
The broad phase is to consider the pairs of rigid bodies possibly collide in the narrow phase and exclude the pairs that do not actually collide. The broad phase uses a bounding shape and a space partitioning to generate an upper bound list of colliding pairs. Various bounding shapes can be considered such as Axis-Aligned Bounding Boxes (AABBs), Oriented Bounding Boxes (OBBs) or simply bounding circle (or sphere). Using these bounding shapes enables the contact detection computationally more manageable by quickly culling out the pairs of not potentially colliding each other. When the AABBs intersect, then the pair is eligible for next narrow phase contact detection, otherwise the pair is excluded.

![Figure 67. Most used bounding shapes. [11]](image1)

![Figure 68. AABBs used for Broad Phase. [12]](image2)

![Figure 69. Broad Phase using AABBs [11]](image3)
Performing detection for all \( n \) pairs in the world make the computational order of \( O(n^2) \) which is a bottleneck. So broad phase uses space partitioning algorithms such as sort and sweep, and bounding volume hierarchies (BVH). Bullet has the implementation for both in \( btAxisSweep \) class and \( btDvbtBroadphase \) class. Dvbt stands for dynamic bounding volume trees which is a type of BVH. The concept of sort and sweep algorithm is to project all of bounding shapes of bodies in the world onto a coordinate axis. This projection will generate intervals for each AABB on the axis, and the algorithm uses the beginning-end intervals to detect intersection. This procedure considerably reduces the number of direct intersection tests [12].

![Figure 70. Sort and sweep algorithm [11]](image)

Any interval as \([b, e]\) and sort all their scalar values of \( b \) and \( e \) ascending is defined in an array. Then by travelling across the array, whenever a \( b \) value is faced, the related interval will be sent to the intersecting intervals list and whenever an \( e \) value is faced, the related interval is removed from the intersecting list. Bullet uses three structures for broad phase which are \( btDbvtBroadphase \), \( btAxisSweep3 \) and \( btSimpleBroadphase \) [10].

The broad phase finds the colliding pairs and passes a list of potentially colliding pairs passed to the narrow phase each timestep. Narrow phase then conducts a series of detailed contact detection using the actual geometry of the bodies and determines if the pair do collide or not. If they do collide, this procedure identifies the contact points and pass the information to the collision solver, where the collision impulse is computed to resolve the collision.
If the distance of two potentially colliding objects is less than a predefined tolerance, then the pair is actually colliding. The major computational challenge is to compute the shortest distance between the two colliding objects. It is generally preferred to work with convex shapes rather than concave shapes due to the relative simplicity of shortest distance computation, while the concave shape can be decomposed of a set of convex shapes.

Separating Axis Theorem (SAT) is one of the widely used algorithms in the narrow phase contact detection. The 3D version of this theorem is separating plane theorem or SAP. Based on SAT, if and only if there is a single axis that the orthographical projections of two convex shapes don’t intersect, then the two shapes are not intersecting each other.

![Figure 71. SAT algorithm](image)

Gilbert-Johnson-Keerthi (GJK) is another narrow phase algorithm, which uses Minkowski difference to determine the distance between two shapes and searches for the closest point to the origin in the resulting shape in an iterative manner. Bullet Physics is also equipped with the implemented GJK algorithm in `btGjkEpaSolver` and `btGjkPairDetector` classes.

Bullet has four major data structures for collision detection: `btCollisionObject` contains the world transform and shape of the object, `btCollisionShape` defines the geometric shape of the collision object such as box, sphere, triangular mesh and convex hull, `btGhostObject` is used for the phenomenon of ghost collision that happens when a surrounding convex hull is used instead of a convex shape, and `btCollisionWorld` is a container of all collision objects [10].

In the narrow phase, Bullet uses the dispatcher iterates over all colliding pairs given from broad phase, compute contact points of each colliding pair, and stores the information in `btPersistentManifold` structure. Choosing the right collision shape for the simulation is important for which Bullet introduces the following algorithm for choosing a proper collision shape [10].
3.4. Rigid body dynamics

Bullet uses the class `btRigidBody` to create moving objects and simulate their 6 degrees of freedom (DOF) motion derived from the `btCollisionObject` class. It identifies the geometric and mechanical properties including shape, transform, friction and restitution and adds other features such as mass, local inertia, velocity, constraints and forces. `btTypedConstraint` class is used for rigid body constraints. The `btDiscreteDynamicsWorld` class includes all of the rigid bodies and constraints and performs the simulation by calling `stepSimulation` function.

There are 3 different type of rigid bodies in Bullet. Dynamic rigid bodies are simply moving or moveable objects. These bodies have constant mass and their motion is updated at each time step of the simulation. Static objects are objects which are not defined to move, while these can still collide with the dynamic objects. Their mass is defined as infinity [10]. Kinematic bodies are objects that can be animated by the user, which can be assigned a infinite mass as well. These can influence on other dynamic rigid bodies’ motion but this interaction is one-way coupled (not mutual) as they are not affected by the dynamic rigid bodies’ motion.
3.5. Motion update

In Bullet, position and orientation of any rigid body is updated at the center of its mass, i.e., used as the basis for the calculation of its local inertia. Calculation of local inertia depends on the shape of rigid body which comes from the `btCollisionShape` class. The `stepSimulation` function iterates over any rigid body in the world and updates its motion in each timestep. Rendering task in Bullet can be leveraged by MotionState function. Using the MotionStates function saves considerable computational effort because rendering is updated only for moving objects and not for bodies that have not moved significantly in the timestep. Moreover, by means of MotionStates one can detect any possible shifts between center of mass transform and rendered objects. Any interpolated body position is performed by MotionState. Interpolation may be often needed for rendering, however if a non-interpolated position of a body is required, it can be obtained directly by `btCollisionObject::getWorldTransform` or `btRigidBody::getCenterOfMassTransform`. MotionState is mainly used when the body is just created and just enters the simulation. During the performing of simulation, `stepSimulation` updates the body coordinates by `btMotionState::setWorldTransform`.

3.6. Setting global variables

Creating a simulation world, applying gravity, motion (transform) initialization, broad phase collision detection that consists of computing AABBs and detecting colliding pairs, narrow phase collision detection that computes contact points and solves collision, and integration or updating motion are all done sequentially in each time step in our dynamic world. The dynamic world is defined in the `btDiscreteDynamicsWorld` class by default, which performs member functions in `stepSimulation` to carry out all above-mentioned steps in each timestep. To put in a nutshell, dynamic world is where the simulation is performed. Following lines shows the code of creating a dynamic world in Bullet Physics for running a simulation:

- The objects are first defined for its initialization, which are `collisionConfiguration` for memory management of collision set up, `overlappingPairCache` for broad phase collision detection, `dispatcher` for narrow phase collision detection, and `solver` for solving contacts and calculating impulses which are objects derived from classes `btDefaultCollisionConfiguration`, `btBroadPhaseInterface`, `btCollisionDispatcher` and `btSequentialImpulseConstraintSolver` respectively.
collisionConfiguration = new btDefaultCollisionConfiguration();
overlappingPairCache = new btBroadPhaseInterface()
dispatcher = new btCollisionDispatcher();
solver = new btSequentialImpulseConstraintSolver();

- Defining a dynamicsWorld as an object of class btDiscreteDynamicsWorld and initializing:
  dynamicsWorld = btDiscreteDynamicsWorld (dispatcher, overlappingPairCache, collisionConfiguration, solver);

- In case importing a world into Bullet’s simulation world is needed, the class btBulletWorldImporter can be used:
  fileLoader = new btBulletWorldImporter (dynamicsWorld );
  fileLoader->loadFile(File Name);
  This file is commonly a .bullet or .obj file exported from Blender.

- Gravity is set:
  dynamicsWorld -> setGravity (btVector3 (0, -9.8, 0));

- Creating rigid bodies is done as below. In Bullet rigid bodies are dynamic bodies of class btRigidBody which is derived from the btCollisionObject class. This means any dynamic body is a collision object and inherits its features such as collision shape, which forms the shape and dimensions of a rigid body. Each rigid body has its own additional features such as mass and inertia. The added rigid body is initialized with its primary transform after definition is added to the world. Collision shapes include Box, Sphere, Cone, Convex Hull or Triangle Mesh, etc. Material properties such as friction and restitution should also be assigned.
  btCollisionShape* colshape = new btSphereShape (btScalar(Radius));
  CollisionShapes.push_back(colShape);
  btTransform startTransform;
  startTransform.setIdentity();
  btScalar mass(value);
btVector3 localInertia (0,0,0);
colShape -> calculateLocalInertia (mass, localInertia);
startTransform.setOrigin(x value,y value,z value);
myMotionState = new btDefaultMotionState(startTransform);
btRigidBody::btRigidBodyConstructionInfo rbInfo(mass, myMotionState,
colShape, localInertia);
btRigidBody* body = new btRigidBody(rbInfo);
dynamicsWorld -> addRigidBody (body);

- The transform for active objects of the world at each time step is done by stepSimulation which performs collision detection and physics simulation.

For (i=0, i<Number of desired time steps, i++)
  dynamicsWorld -> stepSimulation (time step);
The total number of desired time steps is simply the simulation time divided by time step [4], [10].

3.7. Time integration
The main input of stepSimulation function is the time step. Bullet uses an internal fixed time step of 1/60 (= 0.01666) seconds. When a simulation’s time step is smaller than 0.01666 seconds, Bullet will automatically interpolate body movement and put it into MotionStete. The number of simulations to be performed by each stepSimulation call is defined by the user and is passed as the second input to the stepSimulation function. If the explosion simulation is being performed, updated explosion force with time is applied each time step. Then two functions named InternalSingleStepSimulation and synchronizeMotionStates perform the simulation and motion update. Applied forces are cleared after the end of each time step as below:

  applyGravity;
  applyExplosion();
  For (i=0; i<clampedSimulationSteps; i++)
  {
    InternalSingleStepSimulation(Bullet’s fixed time step = 0.01666)
    SynchronizeMotionStates();
  }
  clearForces();[4]
**InternalSingleStepSimulation** is the core of the *stepSimulation* function. In this function which gets Bullet’s fixed time step as an input, first unconstrained motion of any dynamic rigid body is calculated by `predictUnconstrainedMotion()` function. Collision detection is then performed by means of function `performDiscreteCollisionDetection()` function. The solver resolves the contact problem by `solveConstraints(getSolverInfo)` and after-collision velocities and reciprocal applied impulses are calculated. In the next step, after-collision positions are updated and calculated reciprocal impulses are applied as a central impulse and a torque impulse to colliding pair [4]:

```c
integrateTransforms(timeStep);
predictUnconstrainMotion(timeStep);
performDiscreteCollisionDetection();
solveConstraints(getSolverInfo);
integrateTransforms(timeStep);
```

In contact detection procedure, the contact points between two collision shapes are detected. These contact points are preserved in an array called manifold, thus the manifold size is equal to number of contact points. The function iterates over contact points in the manifold and calculates total reciprocal impulse of the colliding pair by means of the calculated reciprocal impulse at any contact point by `btSequentialImpulseConstraintSolver` [10].
## SELECTION GUIDE / PROCEDURE

<table>
<thead>
<tr>
<th>Guide/Procedure</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select demolition method</td>
<td>Choosing between using explosives or wrecking ball</td>
</tr>
<tr>
<td>Complete the preprocessing steps of the bridge model</td>
<td>It is recommended to use Cell Fracture Add-on for discretizing the bridge model. Use Bullet Viewport Constraints Tool for building the constraints.</td>
</tr>
<tr>
<td>Download and install Bridge Demolition Add-on (BDA) into</td>
<td>While BDA is supposed to work with all recent versions of Blender, it is recommended to use Blender 2.79 version, which is used in this project.</td>
</tr>
<tr>
<td>your Blender software as discussed in Section 2.2</td>
<td></td>
</tr>
<tr>
<td>Follow the steps as discussed in Section 2.3 to use BDA</td>
<td>Use the data written onto the text files for post-processing analysis. This includes debris propagation analysis, contact force analysis and explosion force analysis. From the data emitted out by the program, valuable analysis results including the maximum debris distribution radius, maximum contact forces developed in the structure and maximum explosion force applied to the structure during the demolition simulation can be obtained.</td>
</tr>
<tr>
<td>to perform bridge demolition simulation based on your</td>
<td></td>
</tr>
<tr>
<td>selected demolition method</td>
<td></td>
</tr>
<tr>
<td>For the demolition project by explosives, use the</td>
<td>This requires some trial and error procedure.</td>
</tr>
<tr>
<td>obtained parameters from post-processing analysis to</td>
<td></td>
</tr>
<tr>
<td>minimize amounts of explosives and optimize location of</td>
<td></td>
</tr>
<tr>
<td>explosive source points. Debris propagation can be also</td>
<td></td>
</tr>
<tr>
<td>estimated from the result.</td>
<td></td>
</tr>
<tr>
<td>Guide/Procedure</td>
<td>Commentary</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------------</td>
<td>------------------------------------------------</td>
</tr>
<tr>
<td>For the demolition project by wrecking ball, use post-processing data for</td>
<td>This requires some trial and error procedure.</td>
</tr>
<tr>
<td>optimizing the project by estimating the wrecking ball mass and related</td>
<td></td>
</tr>
<tr>
<td>equipment, and to proactively plan hitting points.</td>
<td></td>
</tr>
<tr>
<td>Render the simulation.</td>
<td>You may reference the Blender User Manual [1]</td>
</tr>
<tr>
<td></td>
<td>for more details.</td>
</tr>
</tbody>
</table>
REFERENCES


