Journal of Engineering Sciences, Assiut University, Vol. 39, No 1, pp. 49-71, January 2011

FAST GENERATION OF DEMOLITION SPECIAL EFFECTS ON 3D BUILDINGS

Samia A. Ali [(1)], Khaled F. Hussain [(2)], and Ahmed M. Sayed [(3)]

[(1)]Department of Electrical Engineering, Computer Section, Faculty of Engineering, Assiut University, (Egypt)
e-mail: samia_fattah@yahoo.com
[(2)]Department of Computer Science, Faculty of Computers and Information, Assiut University, (Egypt)
e-mail: khaled.hussain2000@gmail.com
[(3)]Department of Electrical Engineering, Computer Section, Faculty of Engineering, Assiut University, (Egypt)
e-mail: eng_ahmedsayed@hotmail.com

(Received August 8, 2010 – Accepted December 12, 2010)

Creating natural, controllable, and efficient computer-generated demolition effects has become a challenging goal in the area of computer graphics. Such effects are of great use in diverse areas that include video games, computer animation, and special effects in action films. Unfortunately, creating such effects is become complicated and time-consuming due to the demolition's natural complex movements and its related effects that include fragmentation, fog, debris, collapsed walls, and dust. To generate realistic demolition special effects, there are two well-known approaches, manual and destruction simulations. The manual approach relies on choosing appropriate tools and setting its parameters to achieve the desired level of realism. However, this approach requires a long time and effort from the designers. On the other hand, the destruction simulations depend on the nature of the demolished objects. However, this approach is slow due to the complexity of simulating the physical behavior of the destructed objects and the energy produced by the collisions between them. The proposed method is based on the automation of the manual approach which depends on creating the effect through programming a script, list of instructions, to generate believable and compelling demolition effect in a fast manner.

KEYWORDS: *Collision Detection, Computer-generated Effects, Demolition Effect, Fast Generation, Special Effects.*

1. INTRODUCTION

Long time ago, the "Realistic Special Effects" played a great role in cinema industry. The production of such films requires high cost to reach a believable level of realism. So the importance of minimizing the cost of producing the realistic special effects [12, 13] has direct impact on the production of these films through the replacement of dangerous and expensive real effects with computer-generated effects. Nowadays, there are many films contain scenes for explosion and demolition of buildings. The

computer-generated demolition effect of 3D buildings can be performed by using 3ds Max software [1] which is one of the most common software in modeling and animation field. This software has many ready tools that can be used to manually generate the effects [9, 11]. But, this requires long time and effort from designers because it necessitates choosing appropriate tools and adjusting many of their parameters to achieve a believable level of realism. In this paper a method is proposed to minimize the time and effort required to generate the demolition effects on 3D buildings. This is done by automating the manual approach for generating the desired effects. The automation is achieved through programming scripts, list of instructions, by means of MAXScript [1] which is one of the main parts of 3ds Max software.

The demolition effects can be created using real or computer-generated special effects. There are many factors that differentiate between them. The first is the cost factor; the cost of demolishing a real building is much more expensive than the cost of creating computer-generated demolition effects. The second factor is the time required to generate the effect; demolishing a real building requires preparation and execution time which is much longer than the time needed to create computer-generated demolition effects. The third factor is the creativity, which is one of the most important factors. Computer-generated special effects enable designers to experiment and make changes to reach the required level of realism. The designers can regenerate the same effect many times to improve its form. The fourth factor is the scale; it is impossible to generate demolition effect for a large site by using the real approach. The only way to generate such effect is through using the computer-generated approach. Finally, the dangerous factor; without using computer-generated special effects, representing real effects may cause accidents and risks due to human mistakes.

The rest of this paper is organized as follows; Section 2 presents the related work. Section 3 describes the proposed method, and its implementation details are presented in section 4. Experimental results are shown in Section 5. The conclusions of this work are presented in Section 6.

2. RELATED WORK

This section describes the related work of computer-generated demolition effects and the factors influenced its development over time. The methods and techniques used to produce special effects have evolved greatly over the past decade. These works include: the simulation of controlled demolition on 3D building with finite elements, manual computer-generated effects, as well as destruction simulation and the generated dust and debris.

2.1 The Simulation of Controlled Demolition on 3D Building

The most efficient way of a systematic destruction of a building is to use controlled explosives. Sikiwat T., Breidt M., and Hartmann D. [18] are presented problems and solutions for the collapse simulation of large scale complex structures. The demolition of structures using controlled explosives is an efficient technology in particular when it comes to high buildings. However, the collapse induced by the sudden destruction of the structural support requires a careful planning if serious damages are to be avoided. For a long time, this planning has only been based upon the acquired experience and

the knowledge of demolition experts. Various accidents and failures at real world blasting events have been reported in the past, nevertheless demonstrated that empirical approaches are prone to errors. The collapse simulation of large scale complex real world structures on the global level requires advanced and sophisticated multi-body modeling concepts [18].

2.2 Manual Computer-generated Effects

Most of the special effects in modern films use a combination of 2D and 3D techniques. Hasraf Dulull [11] presented 2D technique to manipulate individual rendered images of smoke clouds to create a sense of motion, furthermore, creating falling debris within 3ds Max software [1]. Hasraf Dulull brought all generated elements together in Combustion software [14], and performed the final tweaks necessary to ensure a top-quality end result. With today's applications, 3ds Max and Combustion, Hasraf Dulull worked in multiple planes and used virtual cameras to make his composites and matte paintings come to life. Figure 1 displays the scene before and after generating the effect manually.

Figure 1: The scene before and after generating the effect manually [11].

2.3 Destruction Simulation

There are a large number of existing researches on the destruction simulation of objects [3, 4, 5, 6]. Norton [3] simulated fracturing by calculating the stress inside an object according to a spring model. Smith [4] achieved a computation of fracturing by recalculating the fracture faces of the object. Müller [5] used a finite element method to compute physically-based fracturing of objects in real-time. For representing small debris, the method by Zhang [6] used fine tetrahedral subdivision to create the fragments. However, the computational cost of method [6] to generate extremely fine debris and dust is very high.

2.4 Simulating Destruction and the Generated Dust and Debris

T. Imagire, H. Johan, and T. Nishita [2] are presented a method to simulate destruction of objects caused by their collision and generate dust and debris with various sizes based on the fracture energy. Their simulation [2] is based on an extension of the Distinct Element Method or the Discrete Element Method (DEM) [8] designed to solve the dynamic behavior of objects, called the Extended Distinct Element Method (EDEM) [7]. Each object is represented as a set of elements and nearby elements are connected using springs. Fracture is computed based on the physics-based simulation

of the springs between the elements. For each EDEM element, the fracture energy that breaks it can be computed. Based on this energy, the amount of dust and maximum size of the debris can be determined.

3. PROPOSED METHOD

To generate realistic demolition special effects, there are two well-known approaches, manual and destruction simulations. The manual approach relies on choosing appropriate tools and setting its parameters to achieve the desired level of realism. However, that approach requires a long time and effort from the designers. On the other hand, the destruction simulations depend on the nature of the demolished objects. However, that approach is slow due to the complexity of simulating the physical behavior of the destructed objects and the energy produced by the collisions between them. Most simulation-based methods [2, 3, 4, 5, 6, 9] performed the destruction of objects into relatively large size fragments. However, these methods have a high computational cost. Moreover, the scale of destruction that can be handled by these methods is small. The proposed method is based on the automation of the manual approach which depends on generating the demolition effect in a fast manner through programming a script by means of MAXScript [1]. Simply, MAXScript is a tool that can be used to expand the functionality of 3ds Max software [1]. Also, it can be used to add new features or to customize how 3ds Max behaves by typing in a list of instructions that 3ds Max can execute. The usefulness of the MAXScript lies in its flexibility and simplicity. MAXScript as a language is rich enough to permit the developer to generate effects by programming scripts in 3ds Max software as shown in figure 2.

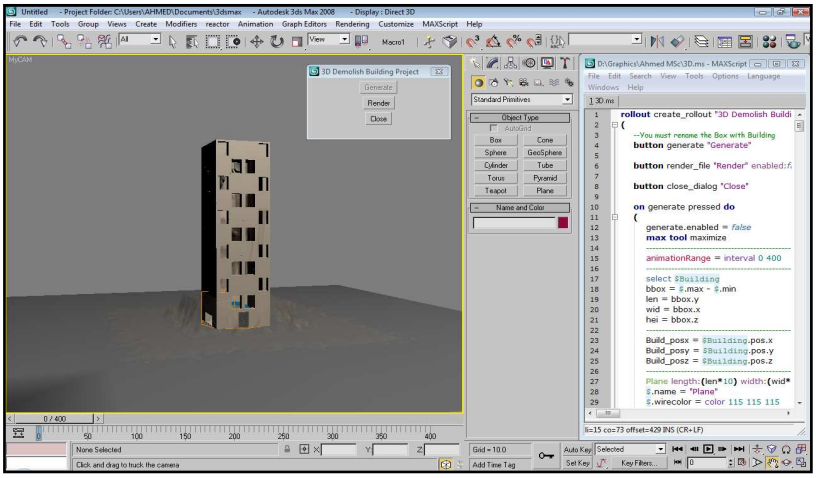

Figure 2: 3ds Max software with MAXScript code.

In order to model the demolition effect on 3D buildings, many different special effects should be combined such as dust, debris, collapsed walls, minor dust generated due to collisions, fragmentation, and fog. These effects are designed by programming scripts. In this paper, the demolition effect is generated on 3D building with ten-story, shown in figure 3, which is taken as a reference model.

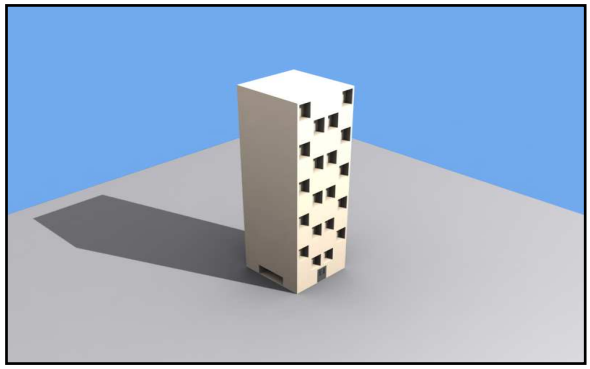

Figure 3: 3D Building to be demolished.

First, the building is split into fragments. These fragments such as debris and collapsed walls are generated offline by creating a single random piece and repeating it with different sizes, positions, and orientations. Fog and dust are generated due to the collisions between the fragments of demolished building. At the end, all scripts of each effect are grouped into one script that generates the demolition effect. For more clarification, the scripts of proposed algorithm for dust and debris effects are presented later.

4. DEMOLITION SPECIAL EFFECTS ON 3D BUILDINGS

The destruction of the building is modeled by combining many different special effects. The following subsections describe these effects by discussing the characteristics of each effect and presenting the algorithm used for its generation. These effects are applied on a 3D building with boundary box that has a length ***len***, width ***wid***, and height ***hei***.

4.1 Dust

This effect is used to generate highland of dust produced from demolition process and cover the fragments of demolished building. The simulation of the dust effect requires a long time because the resulting from demolition effect are billions of granules dust with small sizes. Thus, the proposed method generated this effect by using Plane object, shown in figure 4, in order to reduce the required time.

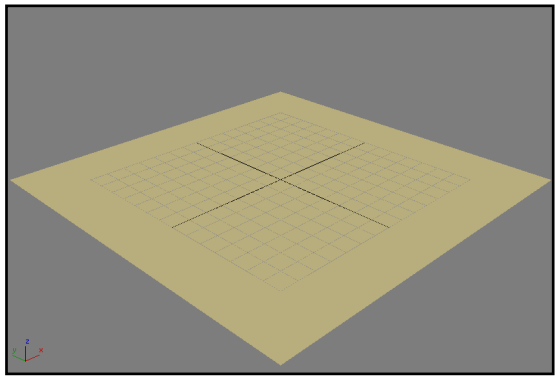

Figure 4: The Plane object.

A plane object is created and named Dust_Plane. The position of this plane is associated with the building position. Due to the dispersion of the demolished fragments during the demolition process, the length ***len_Dust*** and the width ***wid_Dust*** of the Dust_Plane equal, after experiments, five times of ***len*** and ***wid***, respectively. This plane is divided into segments, so the height of each segment can be controlled. The number of rows ***lenseg*** and columns ***widseg*** of segments of the Dust_Plane equal to, after experiments, ***(len_Dust / 10)*** and ***(wid_Dust / 10)***, respectively.

The proposed method can be explained in an example of a plane with five rows ***(r=5)*** and four columns ***(c=4)*** of segments, as shown in figure 5. This plane is converted to Editable Mesh [1], so it can be controlled at vertex, edge, or segment level. Any segment can be identified with two numbers, ***wDust*** and ***(wDust+1)***. Figure 6 displays one segment of the plane. The value of ***wDust*** for any segment, with coordinate ***i*** and ***j***, can be identified according to the following equation:

$$wDust = (j * 2 - 1) + (c * 2) * (i - 1) \quad (1)$$

i \ j	1	2	3	4
1	1 , 2	3 , 4	5 , 6	7 , 8
2	9 , 10	11 , 12	13 , 14	15 , 16
3	17 , 18	19 , 20	21 , 22	23 , 24
4	25 , 26	27 , 28	29 , 30	31 , 32
5	33 , 34	35 , 36	37 , 38	39 , 40

Figure 5: Example of a plane with segments.

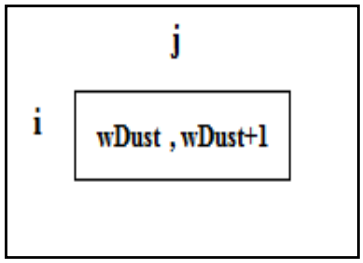

Figure 6: One segment of the plane.

The height of any segment in ***z*** direction ***zDust*** of the Dust_Plane depends on its position in ***x*** and ***y*** direction. Thus, the height of segments inside the center region of the plane, where the presence probability of the demolished fragments increased, is greater than others. The height of other segments is decreased whenever it moves away from the center region. The plane of the Dust_Plane is divided into regions as shown in figure 7. The segments in the same region have the same range of values of their heights. After experimentation, the height ***zDust*** of any segment at region ***w*** can be computed according to the following equation:

$$zDust = \begin{cases} Random\left(\left(\frac{hei}{23+2*(w-1)}\right), \left(\frac{hei}{20+2*(w-1)}\right)\right) & when\ 1 \le w \le 5 \\ Random(-4.0, 2.0) & when\ w = 6 \\ (-4.0) & when\ w = 7 \end{cases} \quad (2)$$

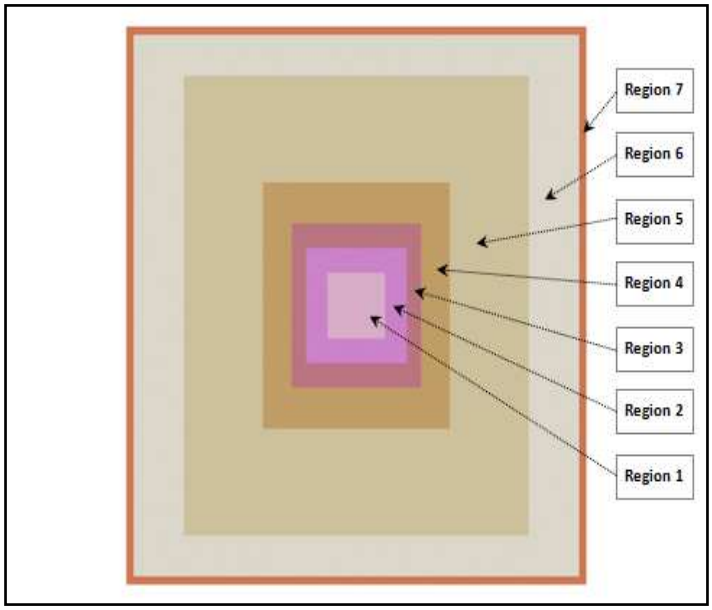

Figure 7: The Dust_Plane is divided into regions.

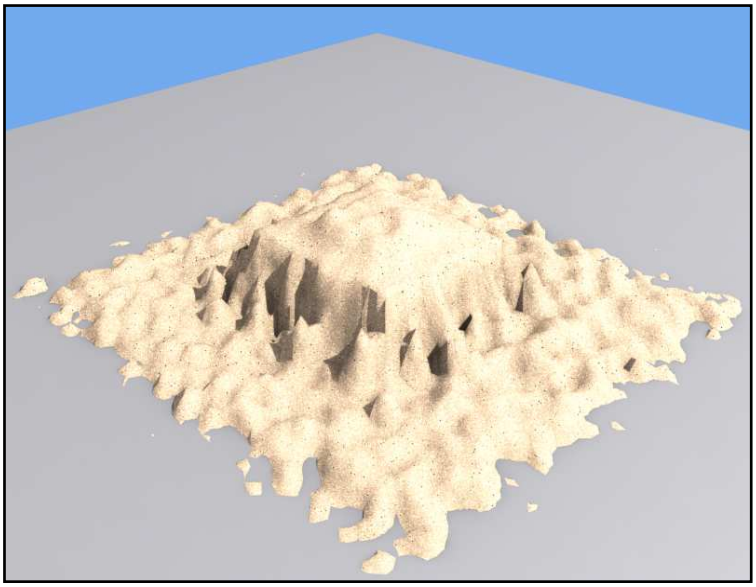

Figure 8: The final dust effect.

The Dust_Plane is covered by a material that has the shape of dust. Figure 8 displays the resulted dust effect. The following instructions specify the proposed algorithm of dust effect:

```
len_Dust = len * 5
wid_Dust = wid * 5
lenseg = len_Dust / 10
widseg = wid_Dust / 10
Plane length:len_Dust width:wid_Dust pos:[Build_posx, Build_posy, Build_posz] isSelected:on
$.name = "Dust_Plane"
$.lengthsegs = lenseg
$.widthsegs = widseg
convertTo $ TriMeshGeometry
for iDust = 1 to lenseg do
(
   for jDust = 1 to widseg do
   (
      r = random 4.0 10
      wDust = (jDust*2-1) + ((iDust-1)*widseg*2)
      if (((iDust > lenseg/2 - lenseg/12) and (iDust < lenseg/2 + lenseg/12)) and ((jDust > widseg/2 -
widseg/12) and (jDust < widseg/2 + widseg/12))) then
            zDust = random (hei/23) (hei/20)
      else if (((iDust > lenseg/2 - lenseg/10) and (iDust < lenseg/2 + lenseg/10)) and ((jDust >
widseg/2 - widseg/10) and (jDust < widseg/2 + widseg/10))) then
            zDust = random (hei/25) (hei/22)
      else if (((iDust > lenseg/2 - lenseg/8) and (iDust < lenseg/2 + lenseg/8)) and ((jDust > widseg/2 -
widseg/8) and (jDust < widseg/2 + widseg/8))) then
            zDust = random (hei/27) (hei/24)
      else if (((iDust > lenseg/2 - lenseg/6) and (iDust < lenseg/2 + lenseg/6)) and ((jDust > widseg/2 -
widseg/6) and (jDust < widseg/2 + widseg/6))) then
            zDust = random (hei/29) (hei/26)
      else if (((iDust > lenseg/2 - lenseg/r) and (iDust < lenseg/2 + lenseg/r)) and ((jDust > widseg/2 -
widseg/r) and (jDust < widseg/2 + widseg/r))) then
            zDust = random (hei/31) (hei/28)
      else if ((iDust > r) and (iDust < lenseg-r) and (jDust > r) and (jDust < widseg-r)) then
            zDust = random 1.0 2
      else if ((iDust == 1) or (iDust == lenseg) or (jDust == 1) or (jDust == widseg)) then
            zDust = -4
      else
            zDust = random -4.0 2
      subobjectLevel = 4
      move $.faces[#{wDust..(wDust+1)}] [0, 0, zDust]
   )
)
```

4.2 Debris

The debris presents the small broken pieces that remain after the demolition of buildings. The debris includes rubble, splinters, and plates. This effect is done offline by generating a single random piece and repeating it with different sizes, positions, and orientations. In order to start the creation of debris effect, any shape object, such as a box or a cylinder, can be used. Figure 9a displays a cylinder object that has been created and named Debris_main. The Debris_main object's size and position have been assigned, after experimentation, with the size of building model by using the following equations:

$$Cyl\_hei = hei / 18 \quad (3)$$

$$Cyl\_rad = (len + wid) / 16 \quad (4)$$

$$Cyl\_pos\_x = building\_pos\_x \quad (5)$$

$$Cyl\_pos\_y = building\_pos\_y \quad (6)$$

$$Cyl\_pos\_z = building\_pos\_z + hei / 2 \quad (7)$$

Where ***Cyl_hei*** and ***Cyl_rad*** are the height and radius, respectively, of the cylinder object. ***Cyl_pos_x***, ***Cyl_pos_y***, and ***Cyl_pos_z*** are the position of the Debris_main on ***x***, ***y***, and ***z*** direction, respectively. ***building_pos_x***, ***building_pos_y***, and ***building_pos_z*** are the position of the building on ***x***, ***y***, and ***z*** direction, respectively. And then many cylinder objects are created and overlapped in random positions with Debris_main object. This can be shown in figure 9b.

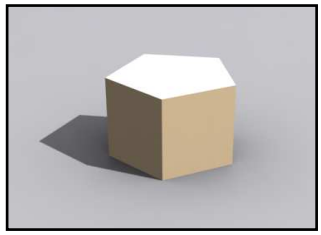

Figure 9a

Debris_main object

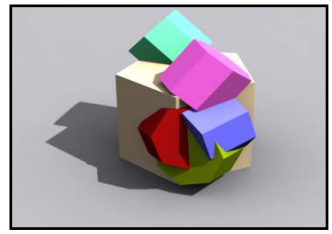

Figure 9b

Cylinders overlapped with Debris_main

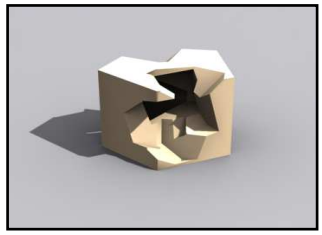

Figure 9c

Debris_main after Subtract operation

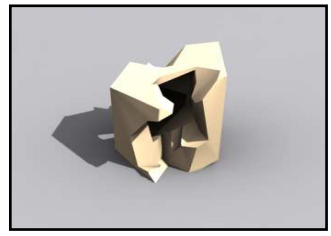

Figure 9d

Debris_main after Noise operation

Figure 9: Steps of generating random shape of debris.

A random shape piece of debris is generated by using the Subtraction operation of the Boolean Compound Object [1]. The subtraction operation is used to subtract each cylinder object from Debris_main object. This is shown in figure 9c. After that, the Debris_main object is repeated to create many objects of debris with different sizes and positions. The number ***k*** of copied objects can be computed, after many of experiments, from the following equation:

$$k = (len + wid + hei) / 1.6 \quad (8)$$

This equation means that the number of copied objects was associated with the size of building. For each copied object, the following steps are executed:

1) A Noise modifier [1] is applied to the object to increase the irregularity of the shape. The Noise modifier parameters are changed randomly to generate different random shapes for each object. Figure 9d displays the object after the Noise modifier applied.
2) The size of each object is changed randomly by the scale equation, which can be calculated according to the following matrix:

$$scale(sx, sy, sz) = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (9)$$

Where ***sx***, ***sy***, and ***sz*** have the same value varies randomly between ***(0.2)*** and ***(0.8)*** for each copied object.

3) Each object is rotated randomly according to the following equation:

$$r(a, x, y, z) = \begin{bmatrix} x^2(1-\cos a)+\cos a & xy(1-\cos a)-z*\sin a & xz(1-\cos a)+y*\sin a & 0 \\ yx(1-\cos a)+z*\sin a & y^2(1-\cos a)+\cos a & yz(1-\cos a)-x*\sin a & 0 \\ zx(1-\cos a)-y*\sin a & zy(1-\cos a)+x*\sin a & z^2(1-\cos a)+\cos a & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (10)$$

Where ***r(a, x, y, z)*** is the rotate value in ***x***, ***y***, and ***z*** direction with angle ***a*** that varies randomly for each object.

4) The area ***a_dib*** where the debris spreads randomly equals, after experimentation, ***1.7*** times of the area of demolished building. The position of each object is changed randomly in ***x*** and ***y*** direction within the area ***a_dib***.
5) Each copied object is covered by noise material that can be look like a broken surface. This can be shown in figure 10.

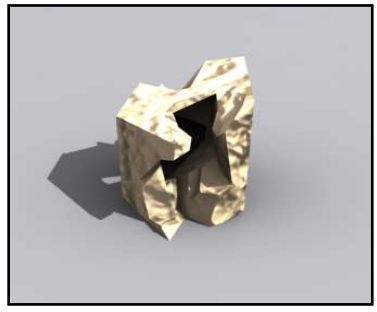

Figure 10: Debris object covered by noise material.

The above algorithm is used to generate various types of debris that includes:

- **Rubble:** broken stones of the building. This can be shown in figure 11.
- **Splinters:** small needle-like pieces that have separated from larger broken windows of the building. This can be shown in figure 12.
- **Plates:** flat thin pieces of broken metals of the building. This can be shown in figure 13.

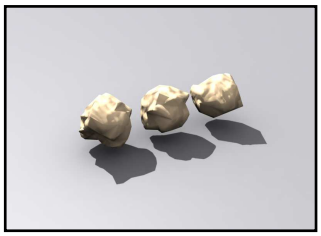

Figure 11: The shape of Rubble.

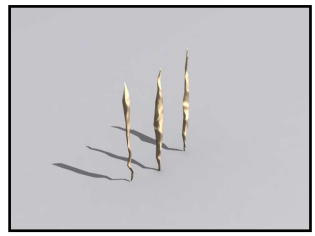

Figure 12: The shape of Splinter.

Figure 13: The shape of Plate.

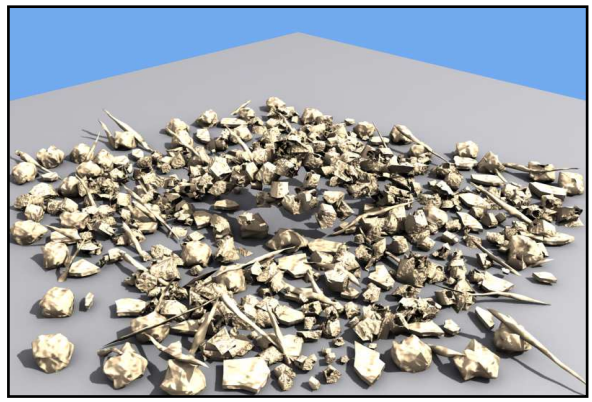

Figure 14: Debris in random positions.

Figure 14 displays the debris in random position. The following instructions specify the proposed algorithm of debris effect:

```
Rub_x = Build_posx
Rub_y = Build_posy
Rub_z = Build_posz+hei/2
Ch = hei/18
Cr = (len+wid)/16
Cylinder smooth:off heightsegs:1 capsegs:1 sides:5 height:Ch radius:Cr mapcoords:on
pos:[Rub_x,Rub_y,Rub_z] isSelected:on
$.name = "Debris_main"
new_Ch = Ch/2
new_Cr = Cr/2
s_xy = -Cr+2
e_xy = Cr-2
s_z = 2
e_z = Ch-2
new_Cy_xy = random s_xy e_xy
new_Cy_z = random s_z e_z
Cylinder smooth:off heightsegs:1 capsegs:1 sides:7 height:new_Ch radius:new_Cr mapcoords:on
pos:[Rub_x+new_Cy_xy, Rub_y+new_Cy_xy, Rub_z+new_Cy_z] isSelected:on
$.name = "Debris01"
rotate $ (angleaxis 50 [1,1,1])
new_Cy_xy = random s_xy e_xy
new_Cy_z = random s_z e_z
Cylinder smooth:off heightsegs:1 capsegs:1 sides:7 height:new_Ch radius:new_Cr mapcoords:on
```

```
pos:[Rub_x+new_Cy_xy, Rub_y+new_Cy_xy, Rub_z+new_Cy_z] isSelected:on
$.name = "Debris02"
rotate $ (angleaxis 100 [1,1,1])
new_Cy_xy = random s_xy e_xy
new_Cy_z = random s_z e_z
Cylinder smooth:off heightsegs:1 capsegs:1 sides:7 height:new_Ch radius:new_Cr mapcoords:on
pos:[Rub_x+new_Cy_xy, Rub_y+new_Cy_xy, Rub_z+new_Cy_z] isSelected:on
$.name = "Debris03"
rotate $ (angleaxis 150 [1,1,1])
new_Cy_xy = random s_xy e_xy
new_Cy_z = random s_z e_z
Cylinder smooth:off heightsegs:1 capsegs:1 sides:7 height:new_Ch radius:new_Cr mapcoords:on
pos:[Rub_x+new_Cy_xy, Rub_y+new_Cy_xy, Rub_z+new_Cy_z] isSelected:on
$.name = "Debris04"
rotate $ (angleaxis 200 [1,1,1])
new_Cy_xy = random s_xy e_xy
new_Cy_z = random s_z e_z
Cylinder smooth:off heightsegs:1 capsegs:1 sides:7 height:new_Ch radius:new_Cr mapcoords:on
pos:[Rub_x+new_Cy_xy, Rub_y+new_Cy_xy, Rub_z+new_Cy_z] isSelected:on
$.name = "Debris05"
rotate $ (angleaxis 250 [1,1,1])
new_Cy_xy = random s_xy e_xy
new_Cy_z = random s_z e_z
Cylinder smooth:off heightsegs:1 capsegs:1 sides:7 height:new_Ch radius:new_Cr mapcoords:on
pos:[Rub_x+new_Cy_xy, Rub_y+new_Cy_xy, Rub_z+new_Cy_z] isSelected:on
$.name = "Debris06"
rotate $ (angleaxis 300 [1,1,1])
new_Cy_xy = random s_xy e_xy
new_Cy_z = random s_z e_z
Cylinder smooth:off heightsegs:1 capsegs:1 sides:7 height:new_Ch radius:new_Cr mapcoords:on
pos:[Rub_x+new_Cy_xy, Rub_y+new_Cy_xy, Rub_z+new_Cy_z] isSelected:on
$.name = "Debris07"
rotate $ (angleaxis 350 [1,1,1])
select $Debris_main
boolObj.createBooleanObject $
boolObj.SetOperandB $ $Debris01 4 2
boolObj.createBooleanObject $
boolObj.SetOperandB $ $Debris02 4 2
boolObj.createBooleanObject $
boolObj.SetOperandB $ $Debris03 4 2
boolObj.createBooleanObject $
boolObj.SetOperandB $ $Debris04 4 2
boolObj.createBooleanObject $
boolObj.SetOperandB $ $Debris05 4 2
boolObj.createBooleanObject $
boolObj.SetOperandB $ $Debris06 4 2
boolObj.createBooleanObject $
boolObj.SetOperandB $ $Debris07 4 2
select $Debris_main
convertTo $ TriMeshGeometry
        meditMaterials[20].bumpMapEnable = on
meditMaterials[20].bumpMapAmount = 150
meditMaterials[20].bumpMap = Noise ()
meditMaterials[20][#Maps][#Bump__Noise____Noise].size = 10
meditMaterials[20][#Maps][#Bump__Noise____Noise].type = 2
meditMaterials[20][#Maps][#Bump__Noise____Noise].color2 = color 115 115 115
meditMaterials[20].Diffuse = color 208 183 144
```

```
select $Debris_main
$.material = meditMaterials[20]
showTextureMap $Debris_main.material true
frm = animationRange.end / 10
for i = 1 to hei do(
        Deb_main = copy $Debris_main
        select Deb_main
        modPanel.addModToSelection (Noisemodifier ()) ui:on
        Noise_seed = random 1 1000
        $.modifiers[#Noise].seed = Noise_seed
        $.modifiers[#Noise].fractal = on
        Noise_ro = random 0.0 1.0
        $.modifiers[#Noise].roughness = Noise_ro
        Noise_it = random 1.0 10.0
        $.modifiers[#Noise].iterations = Noise_it
        Noise_stx = random 15 25
        Noise_sty = random 15 25
        Noise_stz = random 15 25
        $.modifiers[#Noise].strength = [Noise_stx,Noise_sty,Noise_stz]
                                        scale_rub = random 0.2 0.8
        scale Deb_main [scale_rub, scale_rub, scale_rub]
        Deb_main.wirecolor = $Debris_main.wirecolor
        widx = (float(wid) * 1.6)
        leny = (float(len) * 1.6)
        dr = random 10 100
        pos_x = random (-widx-dr) (widx+dr)
        pos_y = random (-leny-dr) (leny+dr)
        if (pos_y > -len and pos_y < len and pos_x > -wid and pos_x < wid) then(
                if (pos_x >= 0) then pos_x = pos_x+wid else pos_x = pos_x-wid
                if (pos_y >= 0) then pos_y = pos_y+len else pos_y = pos_y-len)
        pos_z = random 4.0 6
        rot_x = random 1.0 360
        rot_y = random 1.0 360
        rot_z = random 1.0 360
        rot_deb = eulerangles rot_x rot_y rot_z
        rotate Deb_main rot_deb
        if (mod i 10 == 0) then frm += 1
        set animate on
                at time animationRange.start Deb_main.visibility = off
                at time frm Deb_main.visibility = on
                at time (frm-1) Deb_main.visibility = off
                at time frm Deb_main.pos = [Build_posx, Build_posy, Build_posz + hei/2 - i/2]
                at time (animationRange.end/2) Deb_main.pos = [Build_posx+pos_x,
Build_posy+pos_y, Build_posz+pos_z]
        set animate off )
```

4.3 Collapsed Walls

The collapsed walls look like big broken walls that remain after the demolition of the building. This effect is done offline by generating a single random piece and repeating it with different sizes, positions, and orientations. The algorithm used for this effect is the same as the algorithm which written for debris effect but box tool is used instead of cylinder tool. Figure 15 displays the steps of generating random shape of collapsed walls. Figure 16 displays the wall object after covered by brick material. Figure 17 displays collapsed walls in random positions.

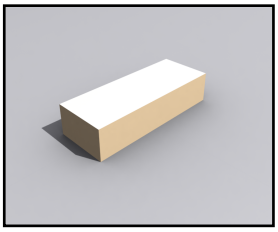

Figure 15a

Wall object

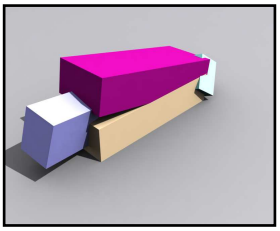

Figure 15b

Boxes overlapped with Wall object

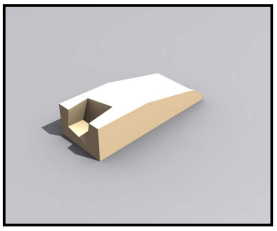

Figure 15c

Wall object after Subtract operation

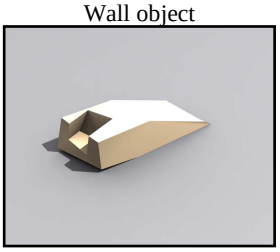

Figure 15d

Wall object after Noise operation

Figure 15: Steps of generating random shape of collapsed walls.

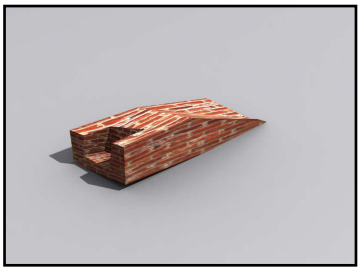

Figure 16: Wall object covered by brick material.

Figure 17: Collapsed walls in random

Figure 18 displays the debris, collapsed walls, and dust effects after demolishing the building.

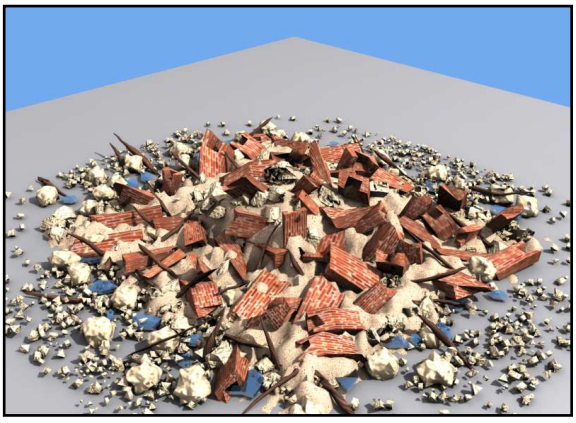

Figure 18: The Debris, Collapsed Walls, and Dust

4.4 Minor Dust generated due to Collisions

During the destruction process, the demolished parts of the building collide with each other and cause minor dust and debris. To detect the collision between two objects, the intersection between all the polygons of these objects must be calculated. The algorithm to compute these intersections requires long time for calculation specially when applied to large number of objects. One of the many ways used to simplify this algorithm is the bounding volume method. The common shapes of bounding volumes are boxes and spheres. The corresponding bounding volumes must be updated when objects perform geometrical transformations. Bounding spheres is a simple task that can be accomplished by updating the coordinates of their centers only. While in bounding boxes, the computation time of the collision detection is longer than the bounding spheres to compare all the coordinates of the vertices of objects to obtain the updated bounding boxes. Thus bounding spheres is the fastest type of collision detection [10]. Figure 19 displays bounding sphere for two collapsed walls.

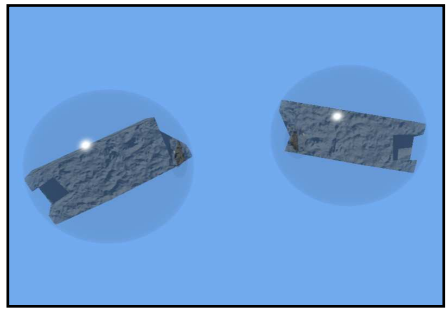

Figure 19: Bounding sphere for two collapsed

Two objects are said to have collided if their bounding spheres overlap. To detect whether a collision has taken place, the radii of the two bounding spheres and the position vectors of the objects must be known. The separation between two bounded sphere objects is compared with the sum of their radii, as shown in figure 20. If the separation between the two objects is less than the sum of their radii, a collision has taken place. If the positions vectors of the centers of these objects ***object01_pos*** and ***object02_pos*** were known, the distance ***dist*** between them can be computed by subtracting the vector ***object02_pos*** from vector ***object01_pos***.

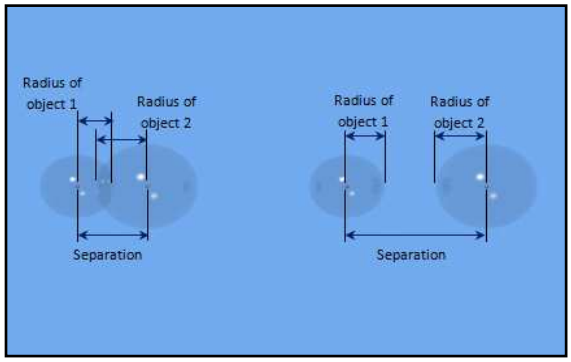

Figure 20: Comparing the separation between two bounded sphere objects with the sum of

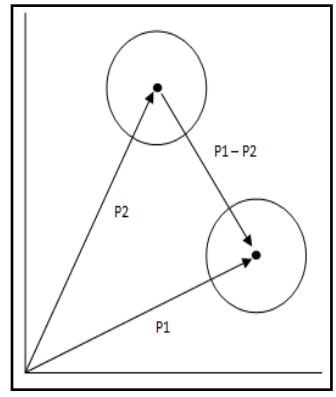

Figure 21: Computing the separation vector between two

Subtracting ***P2*** from ***P1***, as shown in figure 21, gives a vector specifying the distance between the two centers of mass. Thus, the magnitude ***mag*** of the distance vector can be computed. The ***mag*** value is compared with the ***mindist*** which is the sum of the radii of the bounding spheres ***(object01_radius + object02_radius)***. If the distance ***mag*** is less than the sum of the radii ***mindist***, the objects have collided but otherwise, there is no collision.

The above algorithm is slow because it's applied for every pair of particles in the region. The square root in the norm calculation takes about 70 times as long as multiplying floats. To reduce the calculation time of this algorithm, the fast numerical approximations of square root can be used. If there are two numbers ***a*** and ***b***, and ***a*** is greater than ***b***, then a^2 is greater than b^2. The same concept holds here: Instead of comparing the separation distance ***mag*** with the sum of the radii ***mindist***, the square of the separation mag^2 is compared with the square of the sum of the radii $mindist^2$.

If there are ***n*** objects, the above algorithm is applied for each pair of these objects. The number of pairs ***p*** can be calculated from $p = n * (n - 1) / 2$, which requires long calculation time, $O(n^2)$. This calculation time can be minimized as follows:

1- For each pair of objects, if the separation between their centers is greater than a threshold value, this pair is dropped from the calculations because the collision detection is impossible.
2- For each pair of objects, if the separation between their centers is less than the above threshold value, the collision detection is applied for this pair but not in each frame. The frame number that detects the collision depends on the separation between the centers of objects pair. For example, if the separation between two objects is ***S***, the collision detection is applied for this pair after ***fr*** frame, where $fr=S / 20$. This value is deduced after many of experiments. If $fr \leq 1$, the collision detection is applied for this pair after one frame.

Applying this collision detection algorithm results a minor dust shown in figure 22.

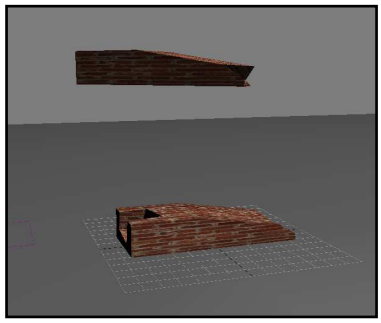

Figure 22a: Collapsed walls before collision.

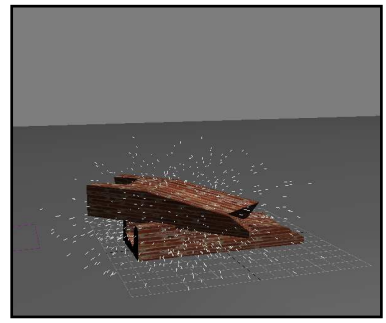

Figure 22b: Collapsed walls after collision.

Figure 22: Minor dust generated due to collision detection between two collapsed walls.

4.5 Fragmentation

This effect is used in the process of demolishing the building. It's generated by splitting the building with many slices planes. Each plane is applied with a Noise modifier [1] and with different position and rotation, in order to increase the irregularity of the

fragments. The building box is fragmented with these slices planes. Figure 23 displays the fragments with different colors of demolished building after splitting it with slices planes. The fragments of the demolished building are moved towards the ground with the effect of the gravity. Figure 24 displays the fragmentation effect.

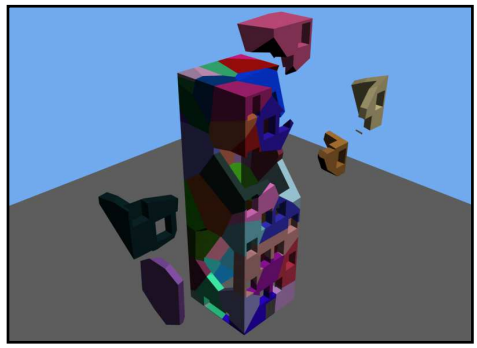

Figure 23: Fragments with different colors of demolished building after splitting it with slices

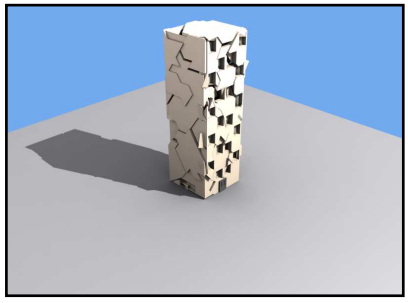

Figure 24a: Fragmentation at frame 55-400

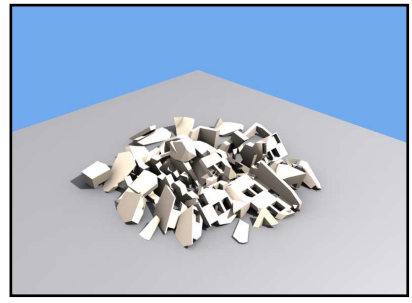

Figure 24b: Fragmentation at frame 400-400.

Figure 24: Fragmentation effect.

4.6 Fog

This effect is used to generate mist of dust, which results during the demolition process. It's generated by using Super Spray tool [1] with help of AfterBurn plug-in [1]. Figure 25 displays the fog effect.

5. EXPERIMENTAL RESULTS

In this section a comparison between the proposed demolition algorithm on 3D buildings and other algorithms [15, 16, 17, 18] on the basis of computation time.

S. Mattern, G. Blankenhorn and K. Schweizerhof [15] are presented computer-aided destruction in order to predict the collapse of complex structure buildings subjected to controlled explosives. Global finite element simulations allow the detection of zones with accumulated damage and structural parts with rigid body like behavior. Combined simulations with flexible finite element part and rigid bodies are compared with the validated finite element analysis. All simulations are performed on eight parallel processors of the Intel® Ithanium® 2-based HP - XC6000 Cluster at the

University Karlsruhe. The average turnaround time for the finite element simulation of one single complete collapse of 9 seconds duration in reality requires approximately 18 hours computation time on the cluster.

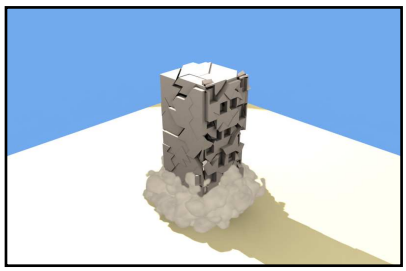

Figure 25a: Fog effect at frame 85-400.

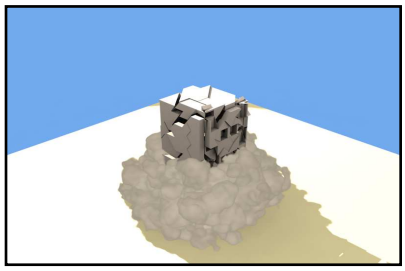

Figure 25b: Fog effect at frame 110-400.

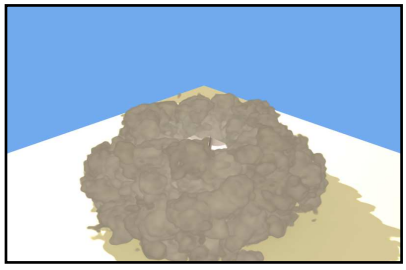

Figure 25c: Fog effect at frame 145-400.

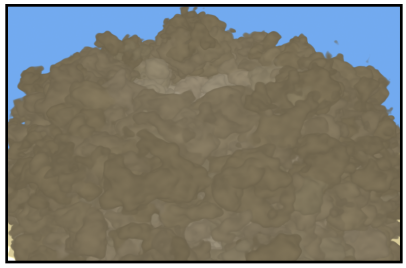

Figure 25d: Fog effect at frame 250-400.

Figure 25: The fog effect.

J. Griffin [16] evaluates the effectiveness of dynamic analysis program in modeling progressive collapse scenarios. He presented a new method of numerical analysis, known as Applied Element Method (AEM). The research is performed to investigate the accuracy of the analysis method contained within the Extreme Loading® for Structures (ELS®) software [16] in modeling progressive collapse of building structures through comparison with the response of each structure during implosion. The analysis was performed on a desktop computer running 64 bit Windows 2003, with a 2.4 GHz Core 2 Duo processor and 4 gigabytes of RAM. It took approximately 55 hours to complete.

Eun-Jin Lee and Sherif El-Tawil [17] are explained the use of virtual reality to present FEMvrml, an interactive virtual environment in which users can interact with and explore the results of finite element simulations. The finite element (FE) method is a popular computational simulation technique which produces information that is proportional to the size of the numerical model. An effective way to develop cost-effective tools is to use readily available open standards software such as the Virtual Reality Markup Language (VRML) [17]. An application, that demonstrates the capabilities of FEMvrml, involves a collapse simulation for an 8-story steel frame building. Data processing time for converting the finite element results into VRML is approximately 8.0 CPU minutes on a machine with a 1.73 GHz Xeon processor and 1 gigabyte of RAM.

Sikiwat T., Breidt M., and Hartmann D. [18] are explained that the simulation of large scale complex real world structures requires sophisticated concepts such that

the simulation model used in the software system can cover the entire dynamic collapse process as well as determine the final debris hill. The physical core of the simulation model is based on a so-called "special multi-body system (special MBS)" that is created adaptively during the simulation process. This concept makes it possible to obtain an efficient and realistic simulation of structural collapse, particularly with regard to the major collapse kinematics. A multi-body simulation model of the reference building is created using 220 rigid bodies. The simulation of the real world duration of 3 seconds requires only approximately 2 hours of calculation time. Hereby, the multi-body model is executed on a machine with a 1.6 GHz Intel Xeon CPU 5110 processor and 2 gigabytes of RAM.

In the proposed algorithm, all programmed scripts of each effect are grouped into one script that generates the demolition effect. This script is installed over a user rollout, shown in figure 26. This rollout allows the user to click on "Generate" button to generate the demolition effect on the selected building. The total time required to execute the proposed demolition algorithm has an average value equals 10 minutes. Then, the user clicks on "Render" button to generate the demolition movie of the building. The total time required to render output movie file with 400 frames has an average value equals 15 hours. This analysis is performed on a laptop running 32 bit Windows Vista, with a 2 GHz Core 2 Duo Intel® processor and 2.5 gigabytes of RAM. The total time required to manually execute the proposed algorithm without using scripts has an average value equals 5 hours without taking the render time in consideration.

Paper Title	System Specification	Simulation Time
Numerical Simulation of Controlled Building Collapse with Finite Elements and Rigid Bodies [15]	Eight parallel processors of the Intel® Ithanium® 2–based HP – XC6000 Cluster	One single complete collapse of 9 seconds duration in reality requires approximately 18 hours computation time
Experimental and Analytical Investigation of Progressive Collapse through Demolition Scenarios and Computer Modeling [16]	Desktop computer, 64 bit Windows 2003, 2.4 GHz Core 2 Duo processor, and 4 GB RAM	Approximately 55 hours
FEMvrml: An Interactive Virtual Environment for Visualization of Finite Element Simulation Results [17]	Machine with a 1.73 GHz Xeon processor and 1 GB RAM	Approximately 8.0 CPU minutes
Collapse Simulations of Large Scale Complex Structures due to Controlled Explosives [18]	Machine with a 1.6 GHz Intel Xeon CPU 5110 processor and 2 GB RAM	3 seconds of real world duration = 2 hours of calculation time
Fast Generation of Demolition Special Effects on 3D Buildings (Proposed algorithm)	Laptop with 32 bit Windows Vista, 2 GHz Core 2 Duo Intel® processor, and 2.5 GB RAM	10 minutes (calculation time) 15 hours (render time)

Table 1: Comparison of simulation time between the proposed algorithm with previous algorithms.

The comparison of simulation time between the proposed algorithm and previous algorithms can be grouped in the following table:

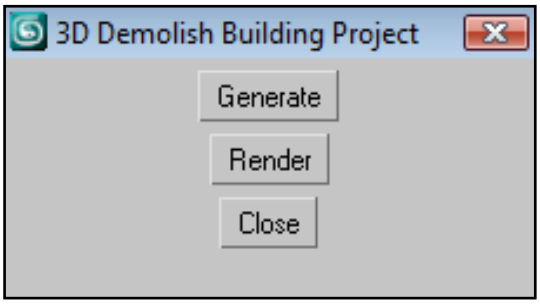

Figure 26: The User Rollout.

As documentation of the real collapse process, the validation is accomplished by a visual comparison of the numerical results to video sequences of the real collapse event. This event was performed on building in 50 High Street, Buffalo, NY City, USA, which was demolished at 6:00AM on May 26, 2007. A visual validation of the simulation is reached via a position of the frames of the video and the visualized results of the simulation at particular points in time during the collapse, as shown in figure 27.

6. CONCLUSIONS

- This paper presented both, manual and destruction simulations, approaches that used to generate realistic demolition special effects.
- The factors that differentiate between computer-generated and real demolishing special effects are illustrated. These factors include cost, time, creativity, scale, and dangerous.
- The related work of computer-generated demolition effects and its factors that have influenced its development over time are described.
- Finally, this paper focused on the proposed method and explained the details of implementing this method for generating demolition special effects on 3D buildings with its related effects that include fragmentation, fog, dust, debris, collapsed walls, and minor dust generated due to collisions.

Finally, it can be concluded that creating computer-generated demolition effects can often become complicated and time consuming due to the demolition's natural complex movements and its related effects. The proposed method depended on automation of the manual approach. It generates the desired effects through programming scripts by means of MAXScript which is one of the main parts of 3ds Max software. Thus, the proposed method minimizes the time and effort to develop believable and compelling demolition effects.

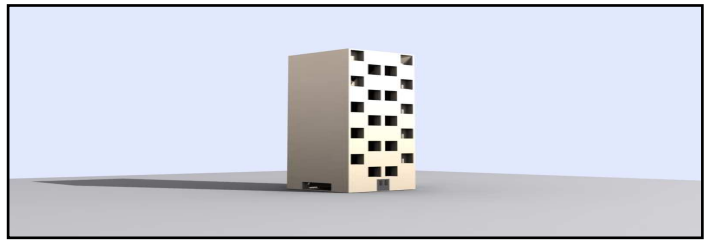

Figure 27a: Frame 001 – 400.

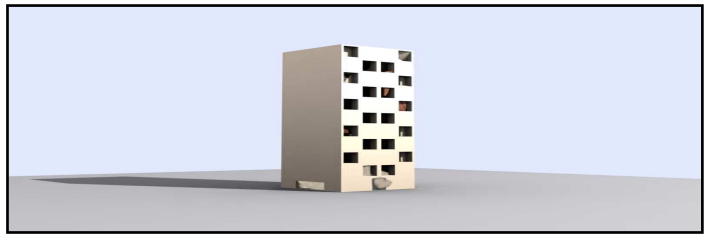

Figure 27b: Frame 053 – 400.

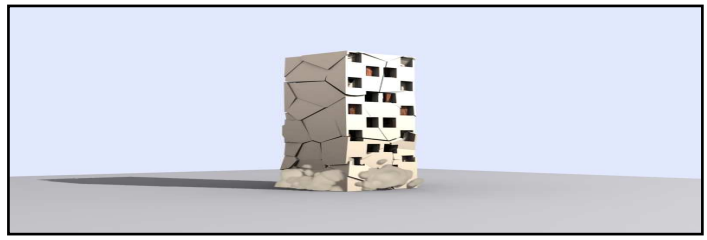

Figure 27c: Frame 064 – 400.

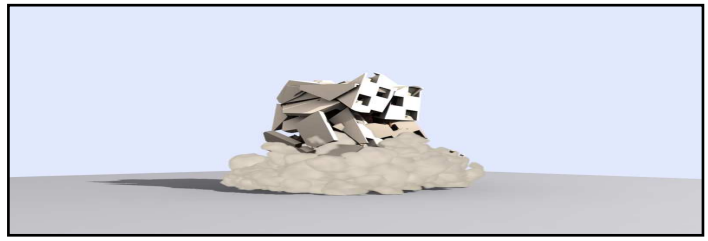

Figure 27d: Frame 097 – 400.

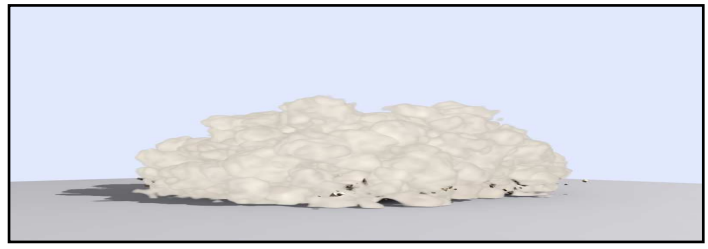

Figure 27e: Frame 144 – 400.

Figure 27: The numerical results to video sequences of the real collapse event.

7. REFERENCES

[1] Kelly L. Murdock, 3ds Max 2008, Bible, 2007.

[2] T. Imagire, H. Johan, and T. Nishita, “A Fast Method for Simulating Destruction and the Generated Dust and Debris,” The Visual Computer Journal, Volume 25, Issue 5-7, pp. 719–727, 2009.

[3] A. Norton, G. Turk, B. Bacon, J. Gerth, and P. Sweeney, “Animation of Fracture by Physical Modeling,” Vis. Comput. 7(4), 210–219, 1991.
[4] J. Smith, A. Witkin, and D. Baraff: “Fast and Controllable Simulation of the Shattering of Brittle Objects,” Comput. Graph. Forum 20(2), 81–91, 2001.
[5] M. Müller, L. McMillan, J. Dorsey, and R. Jagnow, “Real-time Simulation of Deformation and Fracture of Stiff Materials,” In: EUROGRAPHICS 2001 Computer Animation and Simulation Workshop, pp. 27–34, 2001.
[6] N. Zhang, X. Zhou, D. Sha, X. Yuan, K. Tamma, and B. Chen, “Integrating Mesh and Mesh Free Methods for Physics-based Fracture and Debris Cloud Simulation,” In: Symposium on Point-based Graphics, pp. 145–154, 2006.
[7] K. Meguro and M. Hakuno, “Fracture Analyses of Concrete Structures by the Modified Distinct Element Method,” Struct. Eng./Earthquake Eng., JSCE 6(2), 283–294, 1989.
[8] N. Bell, Y. Yu, and P. Mucha, “Particle-based Simulation of Granular Materials,” In: Proceedings of the 2005 ACM SIGGRAPH/ Eurographics Symposium on Computer Animation, pp. 77–86, 2005.
[9] P. Draper, Deconstructing the Elements with 3ds Max, Third Edition, Focal Press, 2009.
[10] Chin-shyurng Fahn and Jui-lung Wang, “Efficient Time-Interrupted and Time-Continuous Collision Detection among Polyhedral Objects in Arbitrary Motion,” Journal of Information Science and Engineering 15, 769-799, 1999.
[11] H. Dulull, The War Zone, Part two, 3ds Max/Combustion, 3D WORLD, pp. 62–65, 2006.
[12] A. Cullen, “Computers and Culture,” Microsoft European Product Development Centre and DCU School of Computer Applications, 2002.
[13] Vesselin I. Shaoulov and Jannick P. Rolland, “Design and Assessment of Compact Optical Systems Towards Special Effects Imaging,” Ph.D, College of Optics and Photonics, University of Central Florida, Orlando, Florida, 2005.
[14] Gary M. Davis, The Focal Easy Guide to Discreet Combustion 4, Focal Press, 2005.
[15] S. Mattern, G. Blankenhorn and K. Schweizerhof, “Numerical Simulation of Controlled Building Collapse with Finite Elements and Rigid Bodies – Case Studies and Validation,” German Research Foundation, 2007.
[16] Joshua Wayne Griffin, “Experimental and Analytical Investigation of Progressive Collapse through Demolition Scenarios and Computer Modeling,” Master of Science, Civil Engineering, North Carolina State University, 2008.
[17] Eun-Jin Lee and Sherif El-Tawil, “FEMvrml: An Interactive Virtual Environment for Visualization of Finite Element Simulation Results,” Advances in Engineering Software, Science Direct, Volume 39, Issue 9, pp. 737-742, September 2008.
[18] Sikiwat T., Breidt M., and Hartmann D., “Collapse Simulations of Large Scale Complex Structures due to Controlled Explosives,” 7th EUROMECH Solid Mechanics Conference, Lisbon, Portugal, September 2009.

التوليد السريع لتأثيرات التهديم الخاصة على بنايات ثلاثية الأبعاد

منذ وقت طويل والتأثيرات الخاصة الواقعية تلعب دوراً كبير فى صناعة السينما من خلال الأفلام التى تحتاج الى قدرة إنتاجية ضخمة للوصول الى الواقعية. ومن أكثر التأثيرات إستخداماً فى صناعة الأفلام وألعاب الفيديو هى تأثيرات التهديم للبنايات. حيث أن التهديم الحقيقى لبناية يحتاج لتكلفة عالية جداً وصعوبة فى التطبيق وقد ينتج عنه مخاطر بسبب الأخطاء البشرية. ومن هنا تظهر أهمية إستبدال مشاهد تهديم البنايات الحقيقية بالتأثيرات الخاصة المُخَلقة باستخدام برامج الحاسب الآلى فى تخفيض تكاليف إنتاج هذه الأفلام بشكل واضح. كما أنه فى بعض البُلدان يتم الإستعانة بالشركات المتخصصة لتهديم البنايات القديمة بإسلوب علمى بحيث لا تتأثر البنايات المجاورة. ويتم تطبيق هذا الإسلوب العلمى بإنشاء محاكاة لتهديم البناية بما يصاحبها من توليد تأثيرات اخرى تتضمن الغبار وضباب الأتربة والحطام والجدران المنهارة. ولكن هذا الإسلوب بطئ لأنه يعتمد على دراسة حركات التهديم المعقدة من الناحية الفيزيائية والهندسية. ويمكن أيضاً توليد هذه التأثيرات بطريقة يدوية بإستخدام الأدوات الجاهزة فى برامج الحاسب الآلى مثل برنامج 3ds Max والذى يعتبر من البرامج الأكثر شيوعا فى مجال الرسومات. ولكن هذا الإسلوب يتطلب مجهود أكبر من المُصَممين لخلق هذه التأثيرات. الطريقة المقترحة من خلال هذا البحث تعتمد على خلق تأثيرات التهديم بطريقة أوتوماتيكية عن طريق برمجة قائمة من الأوامر بواسطة أحد الأجزاء الرئيسيۃ بالبرنامج وهو MAXScript. وهذه الطريقة إستلزمت خبرات فى البرمجة للحصول على تأثيرات تهديم قريبة من المشاهد الواقعية عن طريق تنفيذ قائمة الأوامر المقترحة. فى ضوء ذلك، يتضح أن الطريقة المقترحة تقلل الوقت والجهد للحصول على تأثيرات تهديم واقعية.