2012 IEEE International Conference on Information Science and Technology
Wuhan, Hubei, China; March 23-25, 2012

An Object-Oriented System for Dynamics-Based 3D Cloth Simulation

Hanwen Li and Yi Wan

Abstract—**The dynamics-based 3D cloth simulation has very broad applications. Generally, it involves mathematical modeling, collision detection between objects, self-collision detection for deformable object, and numerical solution of differential equations. As a result, the final simulation software system is usually complex. On the other hand, currently no suitable open-source software systems are publicly available for one to easily use so that his/her idea can be quickly tested. In this paper we first present a data-centric paradigm for dynamics-based simulation, and then propose an object oriented architecture for cloth simulation. In this architecture, we design a common abstract base class for all objects and bounding-box based collision detection, from which we can easily simulate a given type of object by creating a new inherited object class. The inherited class can easily reuse the existing modules in its base class, and hence decrease the number of code lines significantly, thus reducing the complexity and coupling between modules. In particular, when a new algorithm is designed, we can expediently test and verify its effectiveness by directly reuse the other modules without having to reprogram them. The clothing simulation system that we have implemented has fully demonstrated the reusability of the proposed architecture. This architecture also has the potential to be easily used for other types of dynamics-based 3D simulations.**

I. Introduction

THE computer 3D virtual simulation technology has broad application prospects, especially in movie, animation, game, industrial design, and other fields. Computer simulation technology can simulate complex natural phenomena, even the illusion of scenes that not exist in the real world.

In the virtual simulation system, there are geometry-based model, dynamics-based model, and the hybrid model. Because the performance of computer hardware is not rapid enough, early simulations are mostly based on the geometry model, which is simple for implementation. The geometry-based simulation can well fit the hardware performance at that time, but the visual results are not realistic enough. With the development of computer hardware and growing application requirement, the dynamics-based model is gradually used by many researchers. The dynamics-based simulation can gain more realistic results, but its implementation is too complex.

Manuscript received November 1, 2011. This research project is jointly supported by the Fundamental Research Funds for the Central Universities (Project code: lzujbky-2012-k33), and the Program for New Century Excellent Talents in University (Project code: NCET-08-0258).

Hanwen Li and Yi Wan are with the Institute for Signal and Information Processing, School of Information Science & Engineering, Lanzhou University, Lanzhou 730000, China (e-mail: Lhw978@163.com, wanyi@lzu.edu.cn).

In the dynamics-based simulation, we first need to analyze the mechanical forces, and then establish the mathematical model for the objects that will be simulated, such as particle system [1], mass-spring model [2], and so on. After that, the differential equations need to be established according to the dynamics principle. Finally, a stable numerical integration algorithm is designed to solve these differential equations.

Moreover, collisions may happen between two objects, and we need to detect them and make a response. If an object is deformable, we must further process the self-collision detection for it. Because the shape of the object may be irregular, the simulation system is usually complex and hard to meeting the real-time performance requirement. In addition, because of the diversity of objects in the real world, the traditional simulation design methods make the system complex to implement, resulting lower maintainability and reusability. Typically, a full-featured dynamics-based simulation system is composed with tens of thousands of lines of code. In such a case, when a new algorithm needs to be verified, we usually need to reprogram other modules, rather time-consuming and error-prone. On the other hand, currently most of dynamics-based simulation systems publicly available for one are not designed based on the object-oriented thinking, thus no reusability at all [3]-[8].

In this paper, we first present a data-centric complete solution for dynamics-based simulation, including the model source, multiple system mode, offline key frames, and so on. Then a reusable architecture is proposed. Based on the object-oriented design idea, various simulation objects are abstracted to an abstract base class, which includes a number of common operations for all objects. To simulate a given type of object, we only need to create a new class derived from the existing base class. Because most of the functions can be inherited from the existing class, the new class only needs to implement few specialized functions for the given object, thus easy to create and maintain. Similarly, we design the bounding-box based collision detection modules in the same way. By doing this, we can obtain higher reusability for simulating other types of object or for verifying new algorithms by reusing other existing modules, thus dramatically reducing the amount of code lines and redundant functions. Based on this proposed architecture, we have implemented the clothing simulation system, further demonstrated the effectiveness of this architecture.

The rest of the paper is organized as follows. In Section II we review the related knowledge and background for common dynamics-based simulation system. Then the overall solution and architecture are given in Section III. In Section

IV we describe the proposed object-oriented architecture for model object, while in Section V we describe the proposed bounding-box based collision detection architecture. Then the simulation results are presented in Section VI. In Section VII the conclusions are drawn finally.

II. Related Knowledge

With the development of computer hardware, the dynamics-based virtual simulation has been widely used in more and more fields, because of its real visual results. In order to obtain more realistic simulation, the amount of the elements composed the model are generally large, and involve more function modules, thus the computational complexity rise dramatically.

As shown in Fig. 1, in the dynamics-based simulation, we need to analyze the forces exerted on the objects and their component elements. For example, a thrown stone is generally carried by gravity, while the falling snow is also subject to the buoyant force of air, wind force, damping force and so on. For the fluid simulation, the force analysis will be more complex. Then we need to establish its corresponding mathematical model for the simulating objects. Generally, the water, fountain and other scene that are composed with a number of particles can be modeled using the particle system [1], while the bomb explosion and rigid body crash use the elastic model for the simulation [9]. Moreover, the deformable objects such as flag, curtain and clothing are usually modeled with the mass-spring model [2]. The proximity of modeling to the reality is a key factor in dynamics-based simulation, directly determining the quality of visual results.

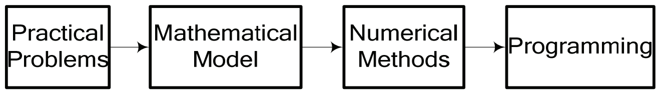


Fig. 1. The simulation procedure.

After the mechanical analysis and object modeling, we need to create the dynamics equations, such as the Newtonian mechanics equations used in deformable object simulation and the Navier-Stokes equations commonly used in fluid simulation [10]. As it is difficult to directly solve these equations using computer in continuous time domain, numerical solution methods are required to approximate the true value of the sampling point [11]. These numerical methods are closely related to convergence speed, system stability and performance.

To obtain more realistic visual results, the simulation system needs to handle collision detection between objects. Otherwise serious penetration phenomenon may happen, badly affecting the fidelity. As the irregular shape of the object or the huge number of elements, the collision detection procedure is one of the most time-consuming parts in simulation. In order to simplify the collision detection and improve simulation performance, many optimization methods have been studied, which are sorted into two categories: the bounding volume hierarchy (BVH) [12] and spatial subdivision [13], [14]. According to the shape of the bounding box, the BVH methods include bounding sphere [15], AABB [16], OBB [17], k-DOP [12] and so on. In the uniform spatial subdivision method, the whole space occupied by objects is divided into many subspaces along the direction of the Cartesian coordinate axis to reduce the number of potential collision detection pairs [13].

For the simulation of deformable objects such as rubber, fabric, clothing, we need to detect whether any self-collisions have happened among its elements of the object. Since the self-collision is a specific collision in essence, there are special algorithms that are studied based on traditional collision detection method [13], [18]. Provot [18] proposed a self-collision detection algorithm based on the normal and curvature of surface with BVH, while Zhang [13] detect self-collision using the uniform spatial subdivision method.

In dynamics-based simulation, the system modules are usually complex, and very large amount of computations are needed, thus it is difficult to maintain, especially in large scenes. To solve these problems, this paper proposes a reusable architecture for dynamics-based simulation system.

III. Overview of the Solution for Animation

A. The Source of Model Data

Firstly, we need to determine the source of model data for simulating. When the shape of object is simple enough, the model data can be directly generated by the software, such as flags, curtains and other objects with regular shape. The snow, rain, smoke and other large scenes can also be randomly generated according to corresponding algorithms. But for other complex objects such as clothing, mannequin, and buildings, it is hard and nonflexible to generate using algorithms. Fortunately, we can first generate them using the third-party modeling tools such as 3DS MAX, MAYA, AUTO CAD and so on, then export them into common model files, and stored as 3DS or OBJ format. Since the format of these types of model file is open-source, we can design a special class for loading the model data from these model files.

B. The Overall of System Framework

As a whole, the simulation system can be divided into two modules: the dynamics-based simulation and the real-time display. The position information of object in each frame is a bridge connecting the two parts, as shown in Fig. 2.

The simulation module is the main part of the system, which is responsible for loading the model data and generating animation frames, while the real-time display module renders timely the points, edges, faces in the key frames on the screen terminal using the interface that the OpenGL or DirectX graphics libraries have provided.

C. The Data-Centric Modules

Since the data is of durable and easy sharing characteristics, the data-centric design has a lot of important applications in business management, database, software engineering, and so

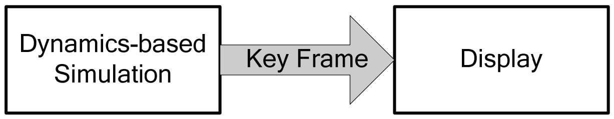


Fig. 2. The system framework.

on.

As we have mentioned, the dynamics-based simulation system usually concerns many modules, such as force calculation, collision detection, self-collision detection, numerical integration, display, and so on. In order to reduce the coupling between these modules, and further improve reusability of the system, we also introduce the data-centric design in our solution for simulation, as shown in Fig. 3.

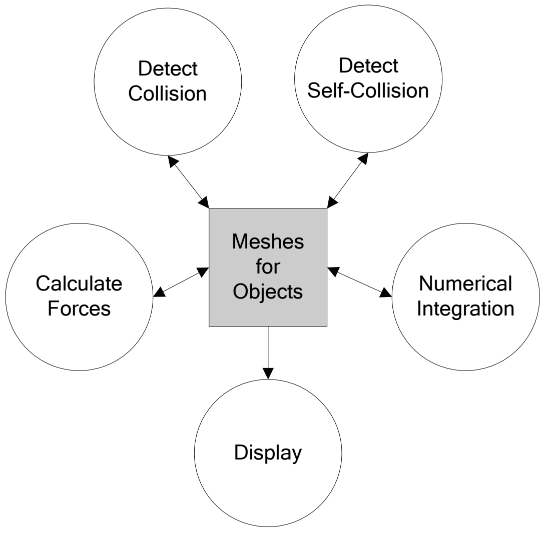


Fig. 3. The data-centric modules.

The meshes for objects are the center of system, by which most of modules around it can exchange information independently, thus reducing the coupling.

D. The Simulation Process

In essence, the dynamics-based simulation is actually to generate a serial of key frames according to the time step. If the system can generate more than 25 frames per second, it will meet the requirements for real-time simulation and display.

In the simulation procedure for generating each animation frame, we must calculate the forces for each object, and then resolve the dynamic equations numerically, then update the position and velocity, as the initial state for generating the next frames. A specific process is shown in Fig. 4.

For the deformable objects, we need to detect self-collision and process the over-stretch phenomenon [2], and then update the position and velocity for each vertex that composed of the object.

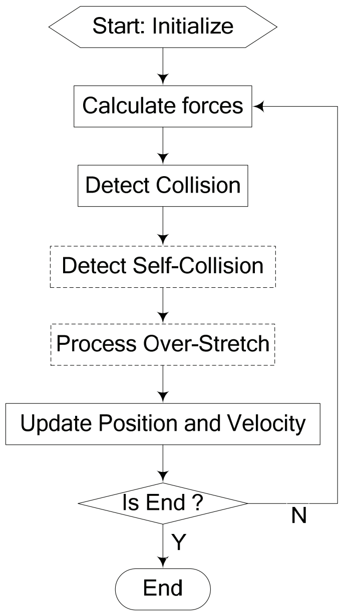


Fig. 4. The dynamics-based simulation process.

E. The System Mode

The system provides three different kind of modes, including online real-time simulation mode, off-line key frame simulation mode, and off-line key frame display mode, as shown in Fig. 5.

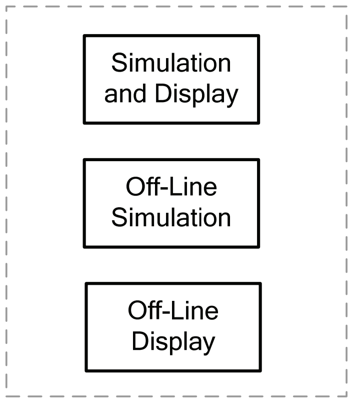


Fig. 5. The system mode.

The online real-time simulation mode is mainly responsible for the real-time simulation and displaying the animation at the same time, which is suitable for simulating the simple objects, such as fabric animation. If the computation is too complex and cannot timely generate animation frames for display, we need to use the off-line key frame simulation mode, which is more appropriate when the scenes are too large or complex to simulate and display in time. When a key frame is generated, it will be stored into temp files. On the contrary, the off-line key frame display mode loads all these key frames from files and playback the animation smoothly.

F. The Animation Frame

We use a list structure to organize the key frames, as shown in Fig. 6. When the current key frame is displaying, the next frame will be prepared.

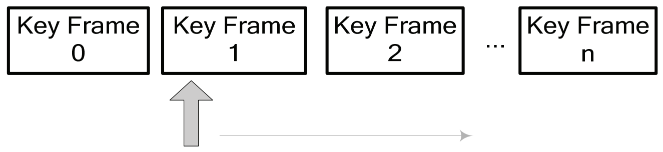


Fig. 6. The list for animation frames.

In each frame, only the position and normal of every vertex are changing, and need to be stored in each frame, especially for the deformable object. In order to further reduce the storage space for key frames, only the matrix with translation, rotation and scale transform information needs to be stored for rigid object.

IV. THE HIERARCHICAL ARCHITECTURE FOR MODEL OBJECT

In this section we describe the key details of hierarchal architecture for model object. The main idea is we first design an abstract class for all objects based on the object-oriented thinking, in which a variety of public operations for simulation are implemented or reserved as virtual function interfaces. Then each specific model object can be derived from the abstract class, and the specialized operations are implemented by means of dynamic binding technology such as virtual function. By doing this, the coupling between modules can be reduced to a large extent, and the degree of reuse for the simulation of other objects is improved greatly.

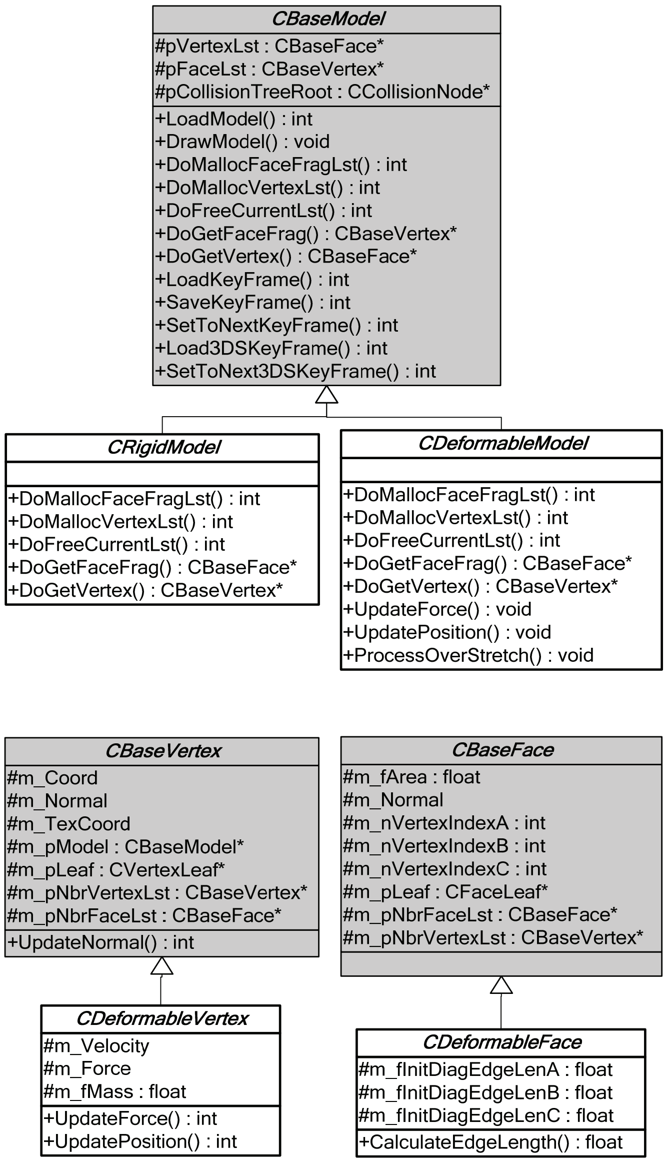
Our architecture mainly supports the simulation for rigid bodies and deformable objects, and can be easily extend to other objects, such as fluid, snow, and fireworks. The inheritance relationships are shown in Fig. 7. In each class we only show the key properties and functions that are necessary for dynamics-based simulation, and other auxiliary functions can be added according to the specific requirements.

Generally, the model in a 3DS format file is mainly composed with triangle patches, while each patch consists of three vertices. Therefore, we first design three base classes respectively for model, patch, and vertex, as shown in Fig. 7. In each base class, a lot of common operations have been implemented. Then we derive two inherited classes for rigid body and deformable object from the base model class. Because the vertices and patches in a deformable object will contain more properties and operations, so we go on deriving two classes for them.

A. Base Classes

1) Base Class for Model

The base class for model mainly stores the lists of vertices and patches, and the corresponding common operations for a given object.



CBaseModel
#pVertexLst : CBaseFace*
#pFaceLst : CBaseVertex*
#pCollisionTreeRoot : CCollisionNode*
+LoadModel() : int
+DrawModel() : void
+DoMallocFaceFragLst() : int
+DoMallocVertexLst() : int
+DoFreeCurrentLst() : int
+DoGetFaceFrag() : CBaseVertex*
+DoGetVertex() : CBaseFace*
+LoadKeyFrame() : int
+SaveKeyFrame() : int
+SetToNextKeyFrame() : int
+Load3DSKeyFrame() : int
+SetToNext3DSKeyFrame() : int
CRigidModel
+DoMallocFaceFragLst() : int
+DoMallocVertexLst() : int
+DoFreeCurrentLst() : int
+DoGetFaceFrag() : CBaseFace*
+DoGetVertex() : CBaseVertex*
CDeformableModel
+DoMallocFaceFragLst() : int
+DoMallocVertexLst() : int
+DoFreeCurrentLst() : int
+DoGetFaceFrag() : CBaseFace*
+DoGetVertex() : CBaseVertex*
+UpdateForce() : void
+UpdatePosition() : void
+ProcessOverStretch() : void
CBaseVertex
#m_Coord
#m_Normal
#m_TexCoord
#m_pModel : CBaseModel*
#m_pLeaf : CVertexLeaf*
#m_pNbrVertexLst : CBaseVertex*
#m_pNbrFaceLst : CBaseFace*
+UpdateNormal() : int
CBaseFace
#m_fArea : float
#m_Normal
#m_nVertexIndexA : int
#m_nVertexIndexB : int
#m_nVertexIndexC : int
#m_pLeaf : CFaceLeaf*
#m_pNbrFaceLst : CBaseFace*
#m_pNbrVertexLst : CBaseVertex*
CDeformableVertex
#m_Velocity
#m_Force
#m_fMass : float
+UpdateForce() : int
+UpdatePosition() : int
CDeformableFace
#m_fInitDiagEdgeLenA : float
#m_fInitDiagEdgeLenB : float
#m_fInitDiagEdgeLenC : float
+CalculateEdgeLength() : float

Fig. 7. The object-oriented architecture for model object.

To facilitate the base class for model can operate its vertex and patch, we only store the address of the two lists with their base class type, and the actual storage memory with derived class type is allocated when initialize the derived model object. So, the base class is abstract and cannot be instantiated. Moreover, the element in these lists must be accessed by means of the virtual functions implemented in derived class for model. By this way, the base class can fulfill more public actions to maximize the module reuse. Furthermore, the base class for model also contains the address of root node of the BVH tree used for collision detection, as shown in Fig. 7.

In the base class for model, other important features are: Loading the model data from 3DS or OBJ files and generating off-line frames, loading off-line frames from temp file, calling the interface in OpenGL or DirectX to display the animation on the screen terminal, calling the interface in collision modules for collision detection, and so on.

2) Base Class for Vertex

The common properties are offered in the base class for a vertex, such as position coordinate, texture coordinate, normal vector, index list of neighboring vertices, and index list of adjacent patches. No other operation is provided in this class, except the function for updating the normal vector.

3) Base Class for Triangle Patch

The base class for triangle patch mainly stores the basic information, such as the indexes for its three corresponding vertices, index list of adjacent patches, normal vector, area, and other information.

B. Rigid Body

Generally, the rigid body may experience the translation, rotation, and scale transformation, Moreover, few operations are needed for a single vertex or patch, so there is no need to derive a new class for the vertex and patch in a rigid body, just deriving a class for rigid body is enough to meet the common needs.

The rigid body class is responsible for allocating, freeing the lists of vertices and patches, and accessing the element in these lists. Besides, some virtual functions for the base class are implemented.

C. Deformable Object

1) Deformable Object Class

Besides of allocating, freeing the lists for vertices and patches, the deformable object class offers other important operations for dynamics-based simulation, such as force calculation, collision detection, numerical solution, over-stretching processing, and so on.

2) Vertex Class for Deformable Object

In this class, a lot of properties of vertex for the deformable object are stored, such as the quality of its own, speed, gravity, damping force, supporting force to the collision, the repulsive force for self-collision, and so on. Besides, other important operations are provided: force calculation, position update, numerical solution, and collision response.

3) Triangle Patch Class for Deformable Object

This class is mainly responsible for calculating the edge length of a triangle patch.

V. The Architecture for Collision Detection

In addition to the numerical solution, the collision detection is another most time-consuming operation in dynamics-based simulation process. Especially in deformable object simulation, self-collision also needs to be detected, thus more time-consuming. This paper mainly describes the hierarchal architecture for collision detection based on the BVH tree. Whereas the spatial-subdivision based collision detection can be packaged into another class.

According to the different shapes of the bounding box, the BVH based methods are divided into many types, such as AABB, OBB, k-DOP and so on. In order to maximize the possibility of module reuse, we also design a object-oriented architecture for these BVH nodes. Again only the necessary properties and functions for collision detection are shown in

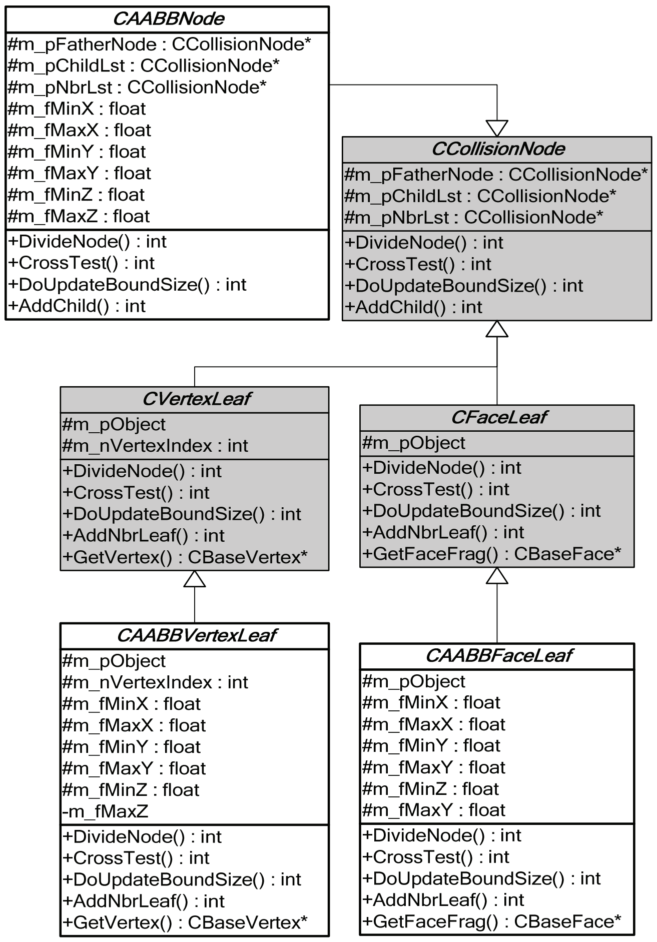


Fig. 8. The object-oriented architecture for BVH-based collision detection.

Fig. 8.

We believe that the leaf node is more special than other node in a BVH tree, because it is directly related to a vertex or patch in a given object. Therefore, the base classes for vertex and patch are derived from the base node class in our architecture.

A. Base Classes

Since the base class has nothing to do with the different shape of the bounding box, so it should be an abstract class and cannot be instantiated, except the difference of base classes from which they derived.

1) Base Class for Node

The major properties are included in this class, such as the address of corresponding model, the father node, the children list, the brothers list, and the interface functions of related operations.

2) Base Class for Vertex Leaf

This class is derived from the base class for node, and stores the pointers of its corresponding vertex and model.

3) Base Class for Patch Leaf

The same as the base class for vertex leaf, this class is also derived from the base class for node, and stores the pointers of its corresponding patch and model.

B. Derived Classes

This class is derived from base class according to the concrete shape of the bounding box. In this section we only illustrate how to derive the AABB node. The nodes with other types of bounding box can be derived similarly.

1) AABB Node

Besides the properties in its base class, this class mainly stores the AABB bounding information, such as the minimum and maximum values in the three Cartesian coordinate axes, as shown in Fig. 8. What is more important, this class is responsible for creating and updating AABB nodes. Actually, the substantial operations for collision detection are also done in this class.

2) AABB Vertex Leaf

This class is derived from the base class for vertex leaf. As the AABB node did, this class also stores the AABB bounding information, total 6 variables with float type, and the same as operations.

3) AABB Patch Leaf

Except that this class is derived from the base class for patch leaf, other properties and operations are similar with the AABB vertex leaf.

VI. Simulation Results

The simulation system based on our proposed architecture is coded using C++ and OpenGL graphics library and runs on a desktop computer with a dual core AMD 2.91GHz processor and 2GB of RAM.

Using the proposed object-oriented architecture, we mainly achieved a fabric and clothing simulation. We first derived the mannequin class from the rigid body class, taking many factors into account such that the mannequin animation is not simulated based on dynamics principle in our simulation system. Alternatively, we directly load the key frames generated by 3DS MAX. Then we derived a class for fabric and clothing respectively. Finally, the collision and self-collision are detected using the AABB tree, which can be easily created with our derived classes for AABB.

First of all, we simulate a piece of flowing fabric which is composed with a grid of m by n vertices. As the initial shape of the fabric is regular, we automatically generate the mesh of the fabric using corresponding program instructions. When the fabric collides with some obstacles in the breeze, a snapshot of the visual simulation is shown in Fig. 9.

We also simulated a clothing animation based on the proposed architecture, in which all the models are loaded from 3DS model files. Among them, the clothing is composed with 2602 vertices, while the mannequin consists of 36978 triangle patches with 85 walk key frames, which are generated with 3DS MAX and exported as 3DS files. The simulation results are shown in Fig. 10.

From the two simulation results, we can see that the proposed architecture is feasible and can well meet dynamics-based simulation for deformable objects such as fabric and clothing animation.

In our simulation system, we have implemented the main modules only necessary for animation, and the amount of

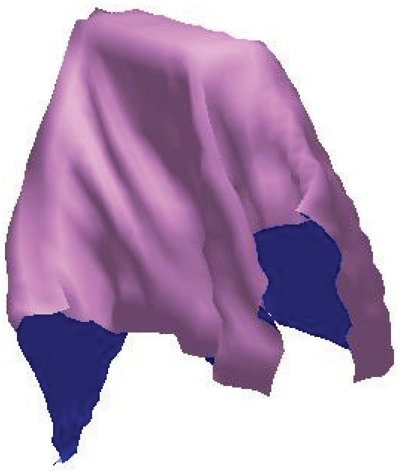


Fig. 9. The simulation result for fabric.

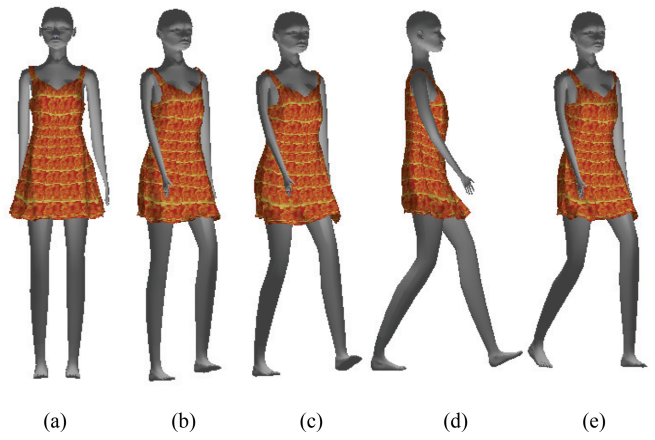


Fig. 10. Some clothing animation results when the human model is walking.

valid code has exceeded more than ten thousand, not including the comments and blank lines. If we use the traditional design mode instead, much more codes would need to be programmed, resulting in the system hardly to maintain or upgrade.

VII. Conclusions

This paper proposes a feasible data-centric solution for dynamics-based simulation, and describes the key details of the design architecture based on the object-oriented thinking for modeling and BVH-based collision detection. This architecture is of good reusability, low coupling between modules, and can reduce the amount of code lines dramatically by eliminating redundant functions. By our architecture, we can easily add or replace with new function modules, which is important for comparing the performance of different algorithms in simulation system for research purposes. That is to say, we can quickly verify the new algorithms without reprogram other modules. Although we only simulated the fabric and clothing animation in our system, the proposed architecture can also be used for other types of objects, only need to derive new classes for the given model, and with a small amount of extension.

References

[1] J. Kruger, P. Kipfer, and R. Westermann, "A particle system for interactive visualization of 3D flows," *IEEE Trans. Visual Comput. Graphics*, vol. 11, no. 6, pp. 744–756, 2005.

[2] X. Provot, "Deformation constraints in a mass-spring model to describe rigid cloth behavior," *Proc. Graphics Interface*, 1995, pp. 147–154.

[3] J. Alex, "SimCloth: Dynamic cloth simulation," Available: http:// sourceforge.net/ projects/simcloth/.

[4] E. Shulman, "Cloth simulation & smoke simulation," Available: http:// clothsim. tripod.com/.

[5] "Freecloth," Available: http://opensourceforce.org/osprojects/20100208/113523.html.

[6] "Opencloth," Available: http://code.google.com/p/opencloth/downloads/list.

[7] "Mosegaards cloth simulation coding tutorial," The Alexandra Institute, Available: http://cg.alexandra.dk/2009/06/02/mosegaards-cloth-simulation-coding-tutorial/.

[8] Paul, "Cloth simulation," Available: http://www.paulsprojects.net/opengl/cloth/ cloth.html.

[9] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer, "Elastically deformable models," *Comput. Graphics*, vol. 21, no. 4, pp. 205–214, July 1987.

[10] T. J. Madden, "Chemical oxygen-iodine layer device simulation using the 3D, unsteady navier-stokes equations," *Conf. DoD High Perform. Computing Mod. Prog. Users Group*, June 2007, pp. 158–166.

[11] L. Verlet, "Computer 'experiments' on classical fluids: I. Thermo dynamical properties of Lennard-Jones molcules," *Phys. Rev.*, vol. 159, no. 1, pp. 98-103, 1967.

[12] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-DOPs," *IEEE Trans. Visual Comput. Graphics*, vol. 4, no. 1, pp. 21-37, Jan. 1998.

[13] D. L. Zhang and M. M. F. Yuen, "Collision detection for clothed human animation," *Proc. 8th Pacific Conf. Comput. Graphics & Appl.*, Hong Kong, 2000, pp. 328-337.

[14] D. L. Zhang and M. M. F. Yuen, "A coherence-based collision detection method for dressed human simulation," *Comput. Graphics Forum*, vol. 21, no. 1, pp. 33-42, 2002.

[15] C. Mendoza and C. O'Sullivan, "Interruptible collision detection for deformable objects," *Computers & Graphics*, vol. 30, no. 3, pp. 432-438, 2006.

[16] G. van den Bergen, "Efficient collision detection of complex deformable models using aabb trees," *J. Graphics Tools*, vol. 2, no. 4, pp. 1-14, 1997.

[17] S. Gottschalk, M. C. Lin, and D. Manocha, "OBB-Tree: A hierarchical structure for rapid interference detection," *Computer Graphics*, pp. 171-180, 1996.

[18] X. Provot, "Collision and self-collision handling in cloth model dedicated to design garments," *Proc. Graphics Interface*, 1997, pp. 177–189.