ISSN:0975-9646

Physics Engine: Deformation Simulation

Abhishek Bhandwaldar
Department of Computer Engineering
Smt. Kashibai Navale College of Engineering
Pune, India

Akshay Gandhi
Department of Computer Engineering
Smt. Kashibai Navale College of Engineering
Pune, India

Mohan Ghule
Department of Computer Engineering
Smt. Kashibai Navale College of Engineering
Pune, India

Nikhil Bansode
Department of Computer Engineering
Smt. Kashibai Navale College of Engineering
Pune, India

***Abstract*— In a game simulation or any software where 3d rendering is concerned, there are various aspect to be considered. For example if we are producing an animated movie, then there are various areas we need to concentrate like the materials , their surface property, lighting, motion, fluid surface simulation etc. Every field is vast and have various physics engines available in market to simulate them. This paper presents algorithm for simulation of deforming bodies, calculating there deformation and generating impulse force.**

I. INTRODUCTION

In today's games or any software related to graphics requires a sophisticated framework for calculating motion. A physics engine is a software framework or software API which is capable of calculating various physical effects such as rotation, gravity effect, ropes etc. and exporting it to an animation software or a game engine to render a completed motion.

During a graphics simulation of a game or any software we have two choices for rendering motions and object interactions, either we use precompiled motion paths or calculate them. Calculating them is in itself a challenge and a programmer needs to deal with the performance issues. This paper primarily focuses on only one aspect which is physics simulation, the deformation simulation. The method mentioned in the paper only considers a cube but can be further extended to encompass various shapes like sphere, cylinder, etc.

The paper is divided into three different sections. The first one presents simple collision detection algorithms. It explains a very simplified approach for implementing the collision detection module for a 3D cube and uses local co-ordinate system rather than world co-ordinate system. The second section introduces collision resolution and force aggregation algorithm and presents various data structures to store and manage various forces: their magnitude and directions. The section deals with spring force calculation and its use in simulating deformation and later on calculating the resultant impulse and applying it. The paper does not deal with the rendering part as it varies from platform to platform.

II. RELATED WORKS

There exist many propriety physics engine specializing in various aspect. Box2d is an open source physics engine which specializes in 2D physics. The engine has been used in many games including Crayon Physics Deluxe, Limbo, Rolando, Fantastic Contraption, Incredibots, Angry birds. It can simulate bodies composed of convex polygons, circles and edges. It can be used for constraint rigid body simulation. The engine is completely written in platform independent C++ and is used on several mobile phones including iPhone, Android phone and Black Berry 10. It was also used in Nintendo and Wii.

Another famous physics engine is Havok Physics engine. The engine specializes in destruction simulation, capable of rendering and calculating physics of more than one object of varied shapes in real time and can still maintain performance. It also specializes in cloth simulation and can be used as a middleware for animation software, or in game engines. This engine was used in more than 500 games aimed at various platforms like the PC, PlayStation, and Xbox. Havok also has been used in products like Autodesk 3ds max, Autodesk Maya, Autodesk Softimage.

PhysX, a proprietary physics engine and middleware SDK by NVidia, specializes in optics and ray tracing. The engine is mainly used for rendering various effects like rigid body dynamics, soft body dynamics, ragdolls and character controllers, vehicle dynamics, volumetric fluid simulation and cloth simulation including tearing and pressurized cloth. The engine is known for its hardware acceleration and its support for CUDA.

The paper presents physics engine which specializes in deformation simulation and hence finally in soft body simulation, by new approach.

III. COLLISION DETECTION

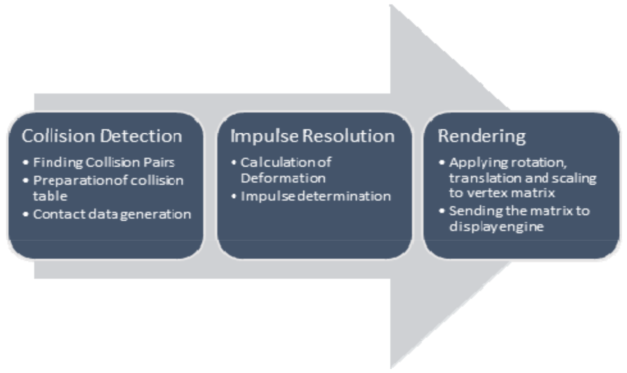
Collision Detection
• Finding Collision Pairs
• Preparation of collision table
• Contact data generation
Impulse Resolution
• Calculation of Deformation
• Impulse determination
Rendering
• Applying rotation, translation and scaling to vertex matrix
• Sending the matrix to display engine

Fig. 1 Physics Engine Pipeline

The engine is divide into various phases. As shown the first phase is collision detection in which we detect collision pairs and prepare collision data. The collision table is further used for contact data generation. Next stage in physics pipeline is impulse calculation and hence calculating velocity, of objects involved in collision. Impulse is calculated by first calculating the deformation and then calculating impulse magnitude. The final phase is rendering. We need to apply final transformations on the vertex data and send it to the render engine. The paper only focuses of on first two phases.

The first phase, namely collision detection is the primary phase of engine. We use vector mathematics for calculation as 3D figures are involved. Firstly we need a specialized data structures to store figures, their various physics related attributes and a table for storing contact data.

To detect if two figures are colliding or not, we need to find the projections of the extreme ends on a normalized vector. Ones we have that we can simply find whether the two figures are colliding or not. Consider we have two cubes cube A and cube B and we want to detect if they are colliding or not. We consider the local coordinate axis rather than the world coordinate axis. We need to find first, normal of two adjacent faces. This two normal will be perpendicular to each other. Consider this two normal as the X and Y axis.

The vector normal to above two vector would be our third axis the Z axis, but we do not need that axis yet. But the face for which that axis is normal is needed. We consider that face only and the two normal as the x axis and y axis. For the second cube we need to find face which is facing the same direction as the normal. We only consider the extreme ends of the faces i.e. the points shown in following fig__. Hence our problem is reduced to 2D collision detection. Now we simply apply the AABB algorithm to see whether the two faces are colliding.

Now we repeat the same process except interchanging the roles of the cubes. The end result tells us whether the cubes are colliding or not in reference to the two axis. Now we repeat same above procedure except now we consider the x axis and z axis. The result returned from this computation tells us whether the cubes are colliding along X and Z axis. The combined result of above two tells us whether the two cubes are actually colliding.

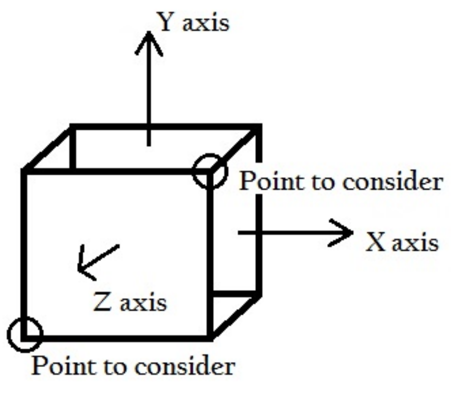

Fig. 2 Cube with its local co-ordinate system

Once we detect there collision we enter the pair in collision table. Next we need collision normal for further calculations. Collision normal is used for calculating the impulse direction and hence the direction in which the cubes will be thrown after the collision. Collision normal differs from figure to figure. As we are only working on cube, collision normal is face normal which is involved in collision. If we have a face and edge collision then collision normal is the face normal of that cube. If we have face-face collision then collision normal of any face would suffice.

IV. Impulse Resolution Through Deformation Calculation

The Next step in our physics engine is impulse resolution. In actual collision bodies deform slightly and hence a force is generated. This force is responsible for pushing the bodies outward. In our scenario we have two cubes which are actual 8 particles per cube connected by spring. Hence every edge of the cube actually represents a spring. In soft-body simulation, the object is collection of particles with each connected through spring to other. Hence to simulate a complete soft body all we need to do is interpolate the presented algorithm for complex bodies.

First we need to simulate a simple spring in our physics engine. A spring in our context is a line with two particles attached at ends and the line acting as an actual spring. To simulate an actual spring we need Hooke's law, given as

$$f = -k\,\Delta l$$

When it comes to three dimensions, we need to generate a force vector rather than a scalar. The corresponding formula for the force is

$$f = -k\,(|d| - l_0)\,\boldsymbol{d}$$

Here vector *d* (non-bold) denotes difference between the two ends of the spring, l_0 natural length of spring and ***d*** (Bold) denotes the normalized vector form of *d*. We use a separate structure known as force registry to store all the force acting on the body. Later on we find the resultant force by using the D'Alembert principal. Now we move on to actual deformation simulation. The first phase has returned us the pairs of collided cubes and the collision table. The collision table consist of the contact data such as the collision normal. The next step would be to first simulate deforming bodies and then apply impulse and calculate separating velocities.

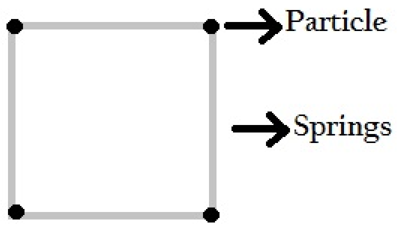

Fig. 3 Face of the cube

When two bodies collide we need to calculate how much compressed or deformed they become. A cube is made up of eight particles, so we will divide particles in two sets. The first set consist of no colliding particle and second set consist of colliding particles. Hence when the cubes collide the two faces involved in collision are affected. The particles on this two faces are put in the colliding particle set and the remaining are put in non-colliding particle set. Next we set the velocity of all particle in colliding set to zero. Hence preventing it from going through each other. Now the non-colliding set particle's velocity is still non-zero and they keep on moving. Hence the shape of the box keeps on changing but we need a decelerating force to stop the particles.

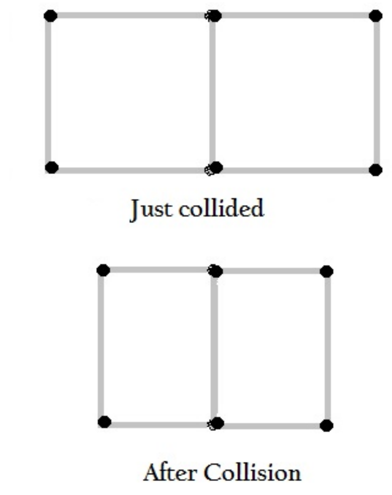

Fig. 4 Deformation during the collision

This force is given by the spring which are its edges. Hence the moment, the collision is detected we change the velocities of particle in the colliding set to zero and we start applying spring force on the particle in non-colliding set. After some time the particles' velocities will become zero, at this point we have a complete collision and we have the springs compressed and storing energy in them. Now we use this spring force and apply it to the non-colliding particle set and start updating the simulation. It appears that the cubes are regaining their shapes and at certain point when the springs between particles gain their natural length we put all the particles from colliding set to non-colliding set and update all particles. Hence the whole cube gets updated and impulse gets applied. The best thing about using spring force is it always exits opposite to the force applied to it, in our case the initial momentum of bodies. Hence the resultant force is in opposite direction to that of collision. This algorithm can be further interpolated for complex figures like sphere of irregular bodies. And hence we can achieve the soft body simulation. We can also assume a cube and surface by just considering cube of same size as the first body and having infinite mass.

V. Future Scope

As mentioned above the method can be further extended for softbody simulation. Softbody simulation is a costly operation and cannot be implemented in real time with other physical effects. By using this algorithm we can implement the softbody simulation in real time and use it in game software and other products demanding real time simulation. Further this method can be extended for simulation of malleable objects. Malleable objects are those objects which deforms after collision and stays in deformed state only.

With slight modification we can use this to make a software which can simulate various complex malleable shapes. This algorithm can be used for software or products demanding effect rather than precision. Another field where this method can be useful is in simulation of destructible mesh objects. Destructible objects involve deformation and fracture generation. Again this algorithm can be used for deformation calculation.

VI. Conclusion

Physics Engine can be used for calculating not only complex motions but also to render various physical effects like rigid body deformation. This paper presents a basic algorithm for simulating deformation of simple bodies.

Acknowledgment

This research was supported by Smt. Kashibai College of Engineering, Sinhgad Technical Education Society, and University of Pune. We also like to express our regards towards Prof. G. R. Shinde, Computer department of SKNCOE, Pune for their continuous support and giving us the useful resources in developing the physics engine.

References

[1] David M., Bourg, Bryan Bywalec, "Physics for Game Developers", O'reilly 2013.

[2] Ian Millington, "Game Physics Development", Moran Kufman publishers, Elsevier 2007.

[3] Wang Xiao-rong, Wang Meng, Li Chun-gui, "Research on Collision Detection Algorithm based on AABB".

[4] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, etc. "Gpu Computing", vol. 96, No.5, May 2008.

[5] Chang Liu, Yue Cao, Xiangi, Quan Xu, Leiting Chen, "Phusis Cloth: A Physics Engine for Real-time Character Cloth Simulation", IEEE 2012.

[6] Yue Cao, Leiting Chen, Chang Liu, Xiao Liang, Hongbin Cai, "Phusis studio: A Real-time Physics Engine for Solid and Fluid Simulation", IEEE 2011.