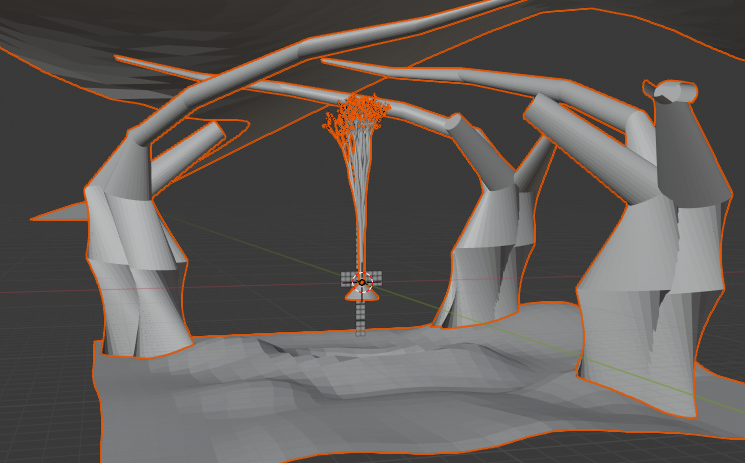
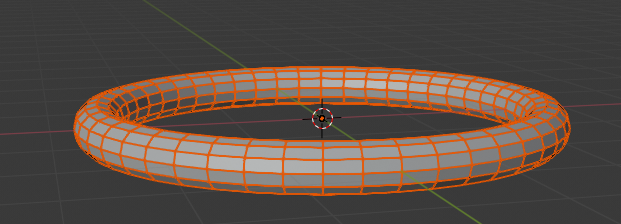
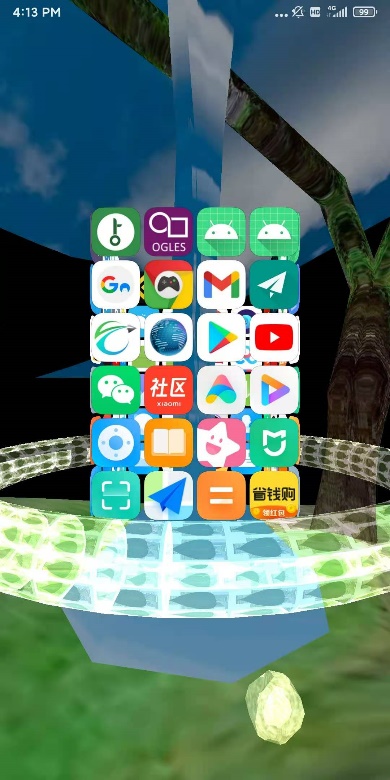
In this document I’ll show two tiny projects finished by 3D graphics rendering method. The first one has been released in public market such as app store of Xiaomi, Huawei, Oppo .

The purpose of doing this software package is to show an usual application with 3D graphics is feasible like any other game applications. To do such a thing, there are three core features to be implemented which are common in any of application frameworks. Also they will be shown in an extreme minimum requirement of resources including Apk size and memory.

Following list is the three core features:

1. Common geometry element to be rendered
2. Common GUI element to reacting the user input.
3. Common GUI Event dispatcher.

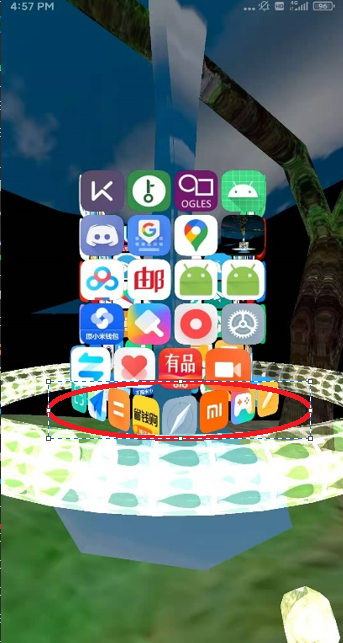
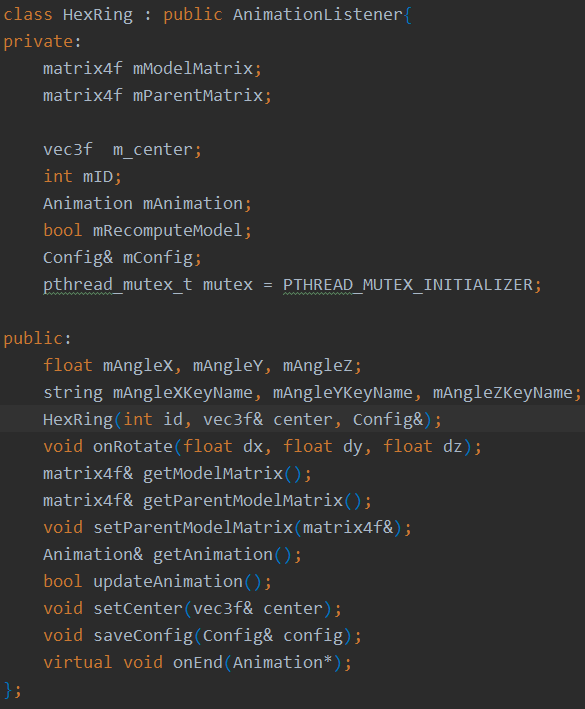
Just like this , here are pictures of final appearance, it’s a normal launcher application of mobile. However it’s basic UI element consists of 3D triangles. For example, the app icon consists of two triangles with 6 vertices. The ring rotates around the application panel is more complicate mesh with bunch of triangles and vertices.



As a tradition of framework design, we need a hierarchy of view. In the first tiny project this part of feature was even omitted. Just used the geometry class as a view object to draw itself. This is an over simplified design. Basically it’s a weird design integrated three level of things in one level. Here is the Quad class for application icon.

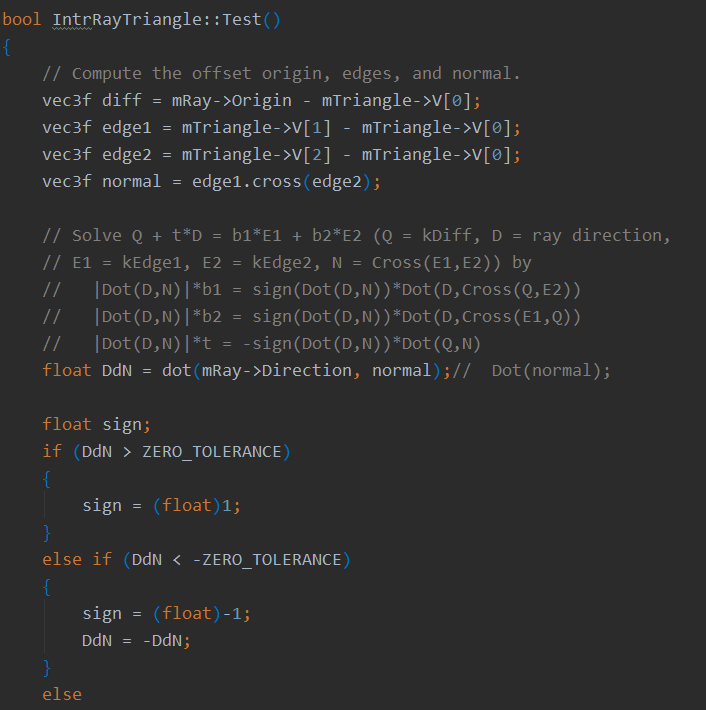


There are still some composite UI elements to be manipulated like the application rings indicated by red circle .

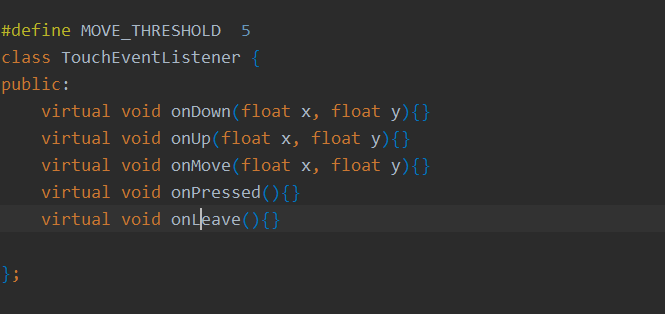
 

So this is about the design of core feature 1. I think it’s straight and clear.

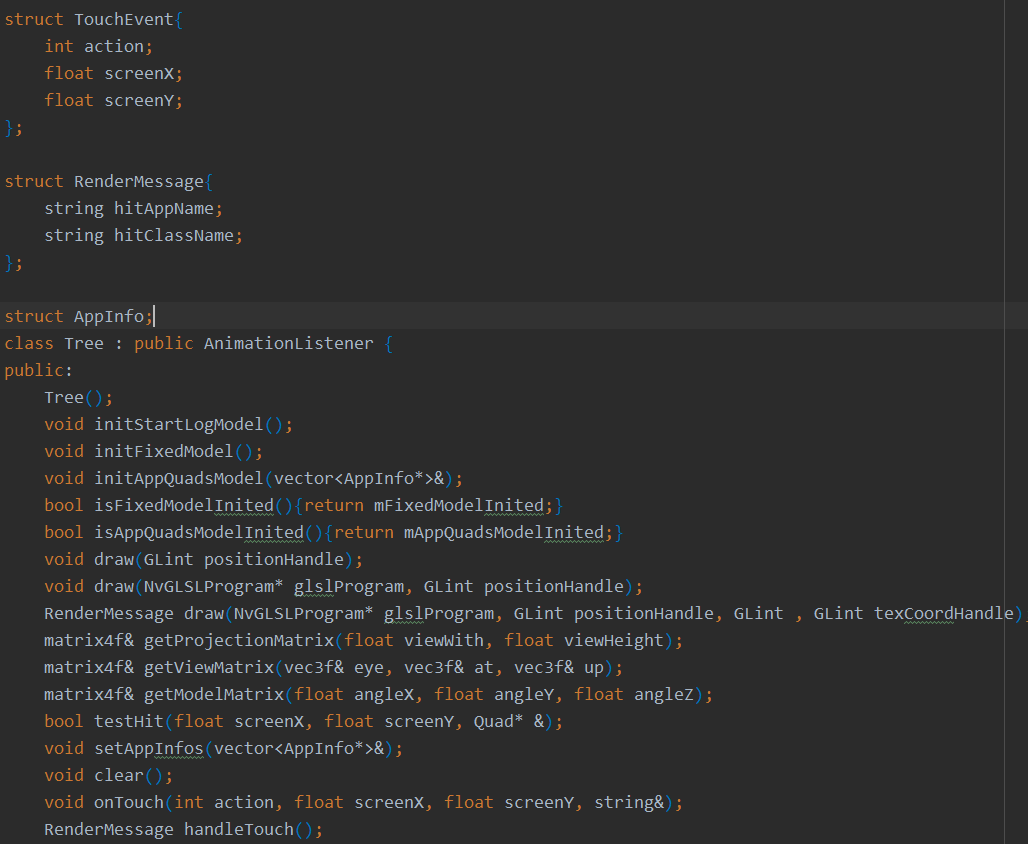
About feature 2. There should be a response when user clicks the application icon or other elements. The reponse is called user event handler. In low level technical details we need a collision detection method to find which app icon does user click. Especially for 3d scene this is not a trivial task. We need process collision between the ray and triangle. Actually it’s relatively easier compared with collision between more complicate geometry.



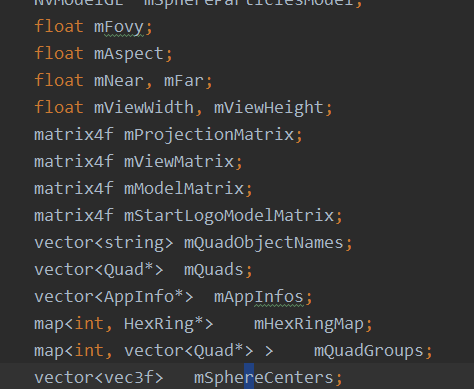
We now can handle the user input event in high level with this low level detection method.



Finally comes to the last feature which is event dispatcher. Just one class handle all of the delivered messages. Messages or events like drawing and touching are processed in the drawing thread. And this extremely tiny framework actually works. It means adding more elements to scene is feasible though there is not a hierarchy for the objects to be rendered.

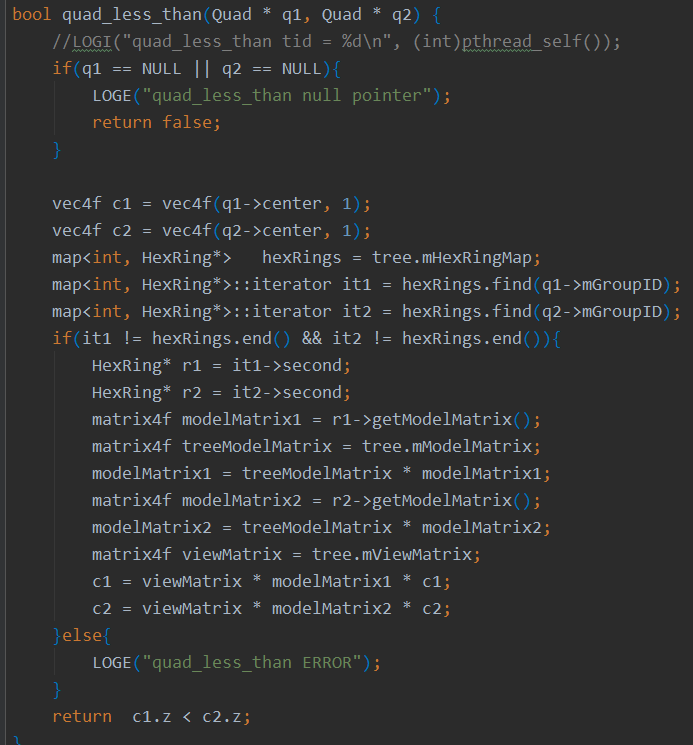


The .obj file format is used for more complicate 3D objects. To model a bunch of icons of which geometry is a quadrilateral it’s necessary to add dynamic data structure for storing these elements. The main reason why it involved this low level design is that the system was lack of hierarchy of 3D Objects.

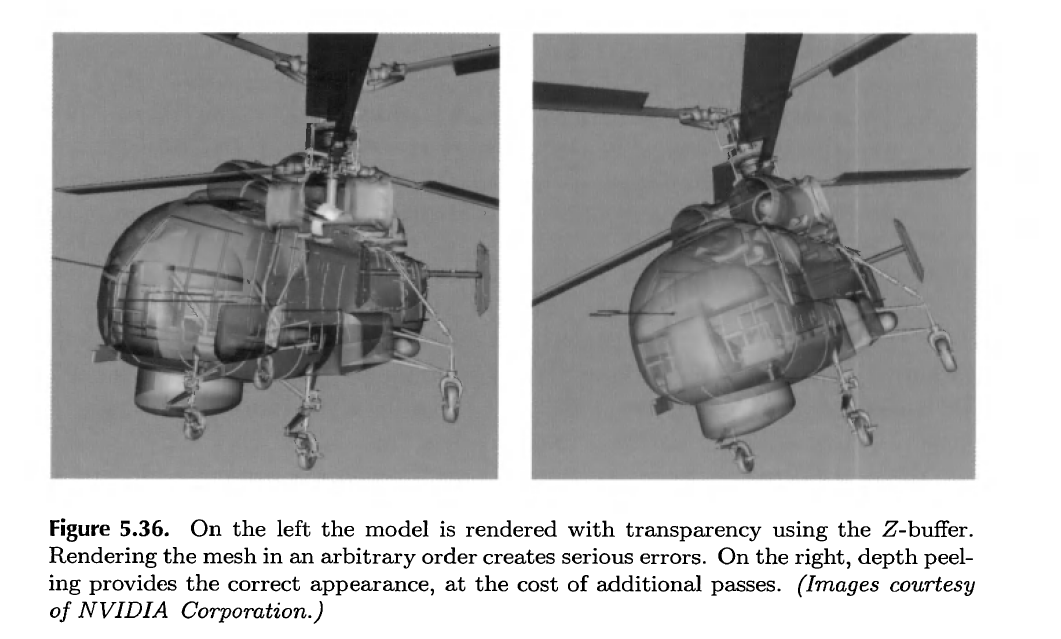


Some of the issues on the implementation phase.

Although we can use the depth test of OpenGL , there are some cases that objects need to get the correct sorted order for rendering. The sorting method is like this :

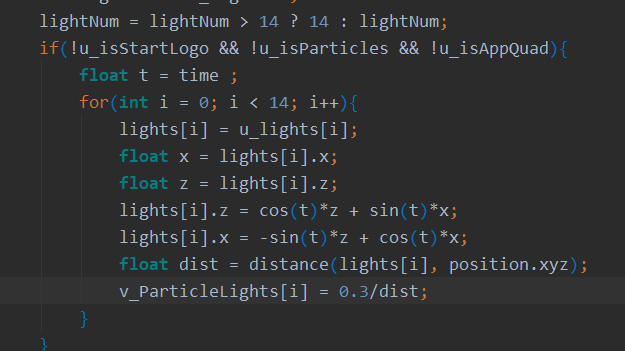


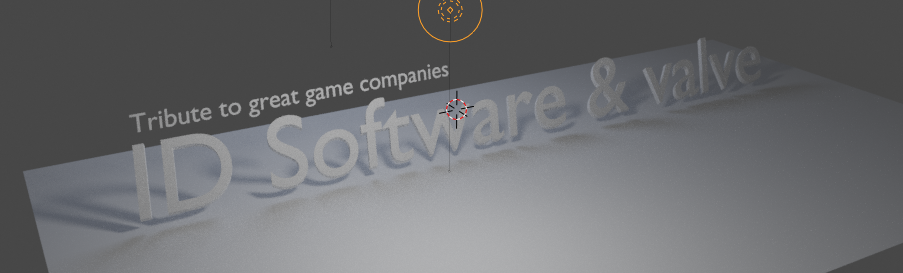
And the issue is about like the following picture described.



About the rendering method

To run the software on different device as much as possible. Set a simple method of rendering like this which is obviously a naïve method based on vertex shader. As the following scene you can see, there is a clearly brightness change of background when the night comes in. Meanwhile the light effect should be observed.

 sdsd



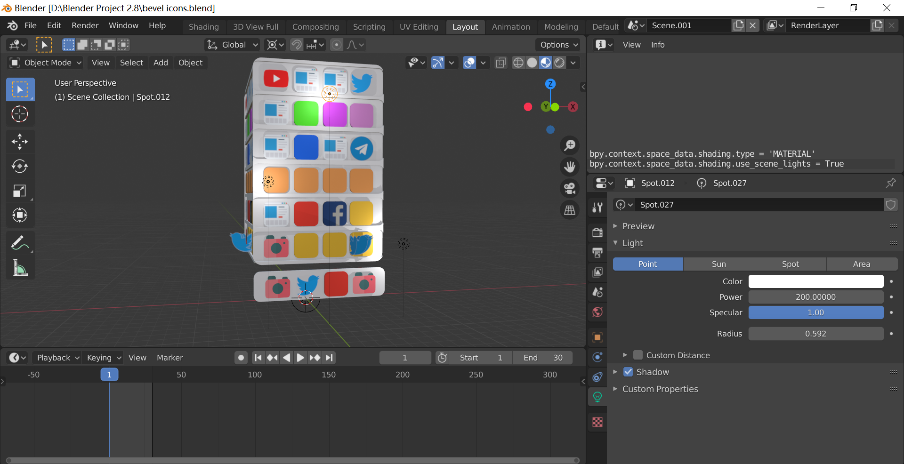
Above figure is the logo showed when software is rebooting. It’s also modeled with 3d mesh.

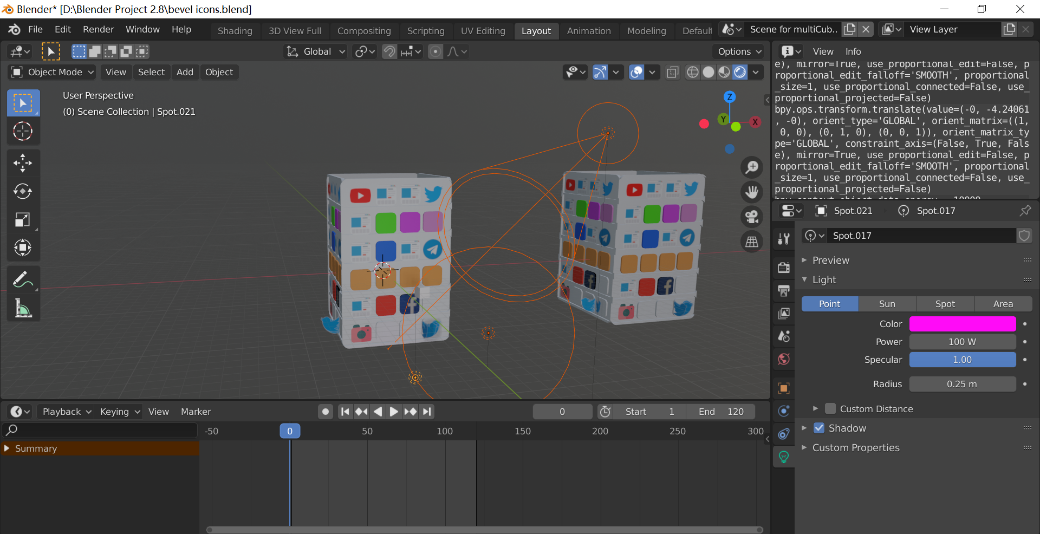
About market feed:

There is a module for estimating the behavior of users. Data shows that children and the young love the 3D graphic elements.

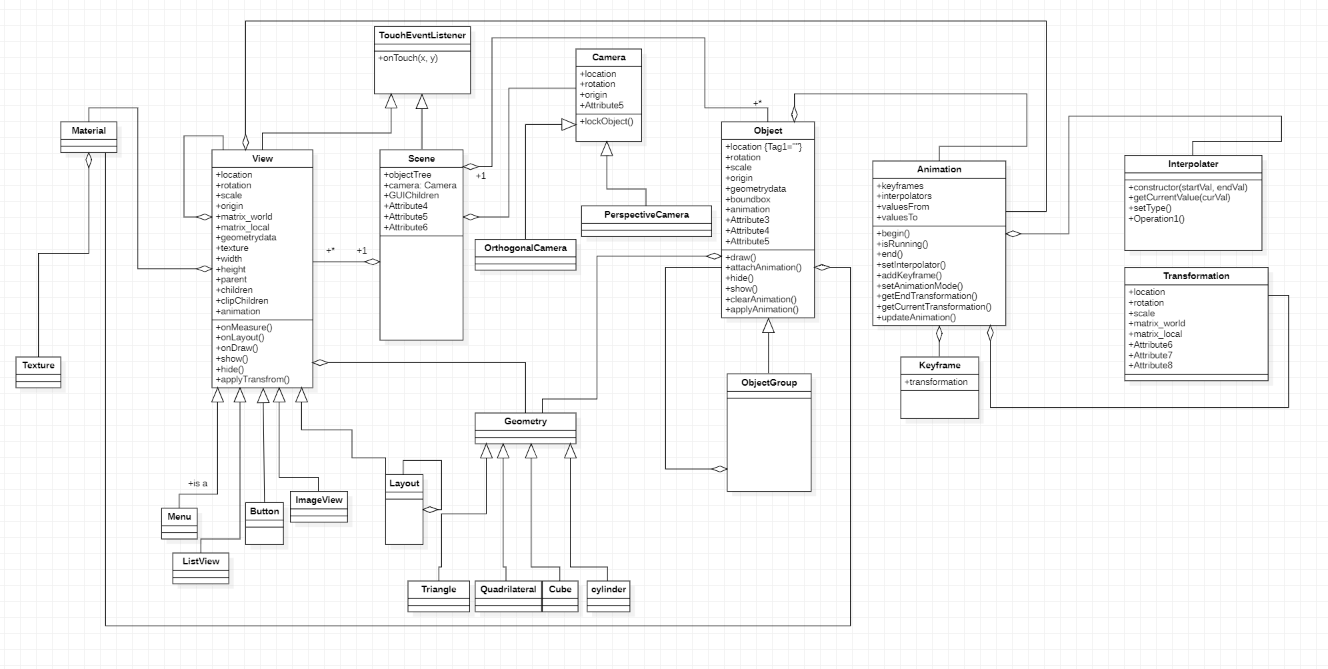
The second project has much more features. It contains object hierarchy, more accurate rendering equations, geometry system, physics simulation, particle system, shadow mapping for drawing shadows, animation system, collision detection for convex objects.

One of the applications is like this. As we can see, blender as a modeling tool was deeply used. Not only the visual toolset is used, also is the python script which can manipulate the objects defined in the phase of modeling.

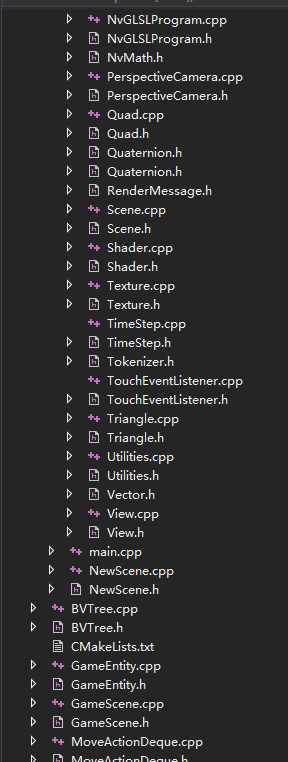
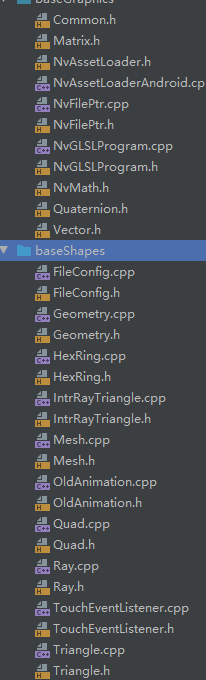
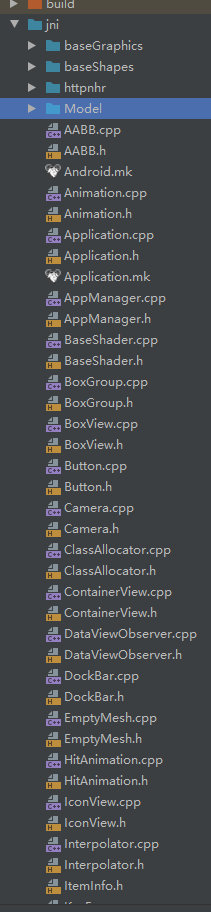




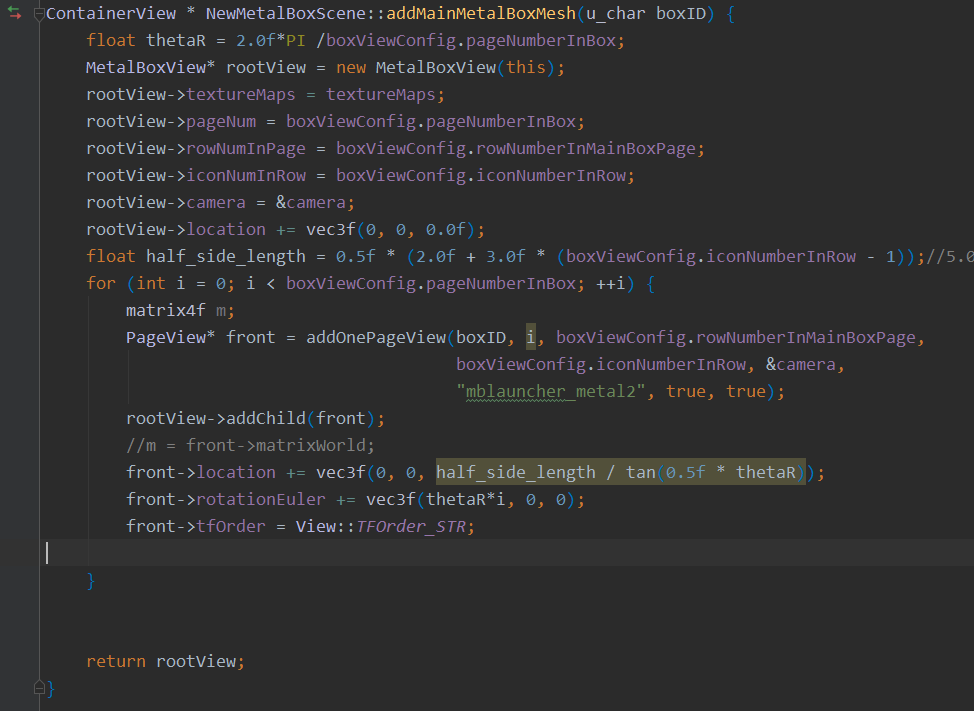
Now the whole system would look like the relations showed by this graph



The code of project would be like



One of main advantages of scaling the framework to such a size is that an application developer can easily write code like the following style



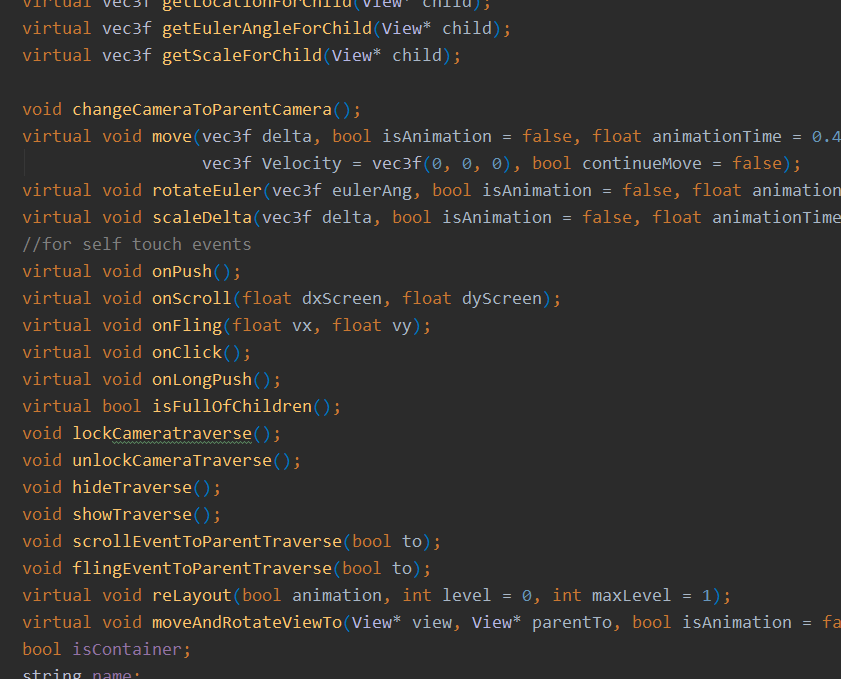
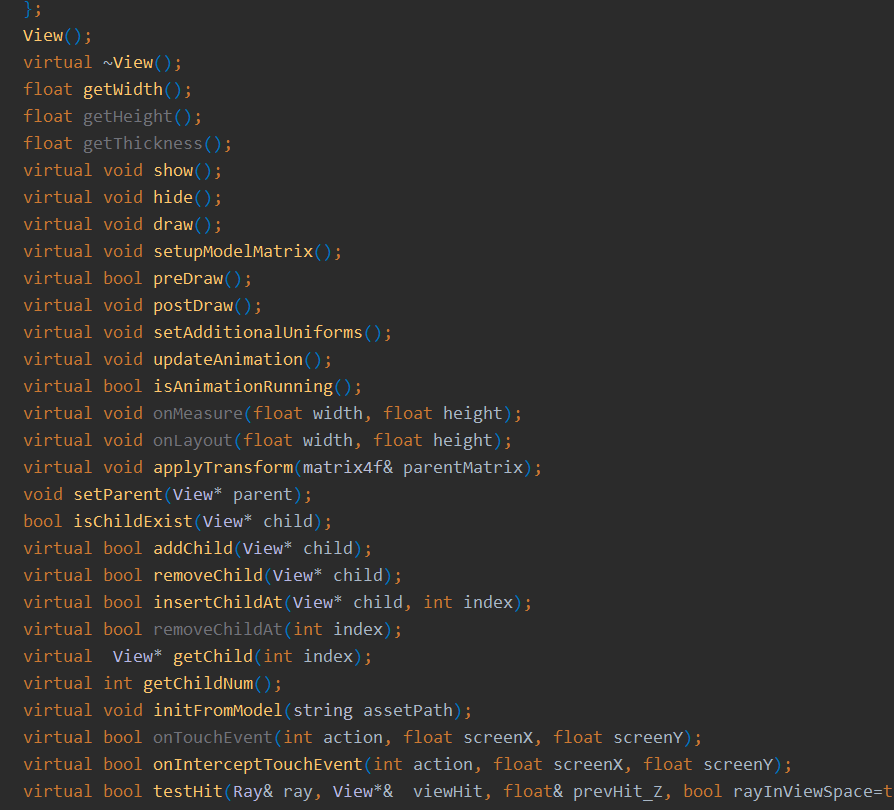
It’s about 20 lines of code an application developer can create a 3D scene with massive objects. All the things that the developer need to do is setting some properties on some of the objects which comes from built-in objects in the framework.



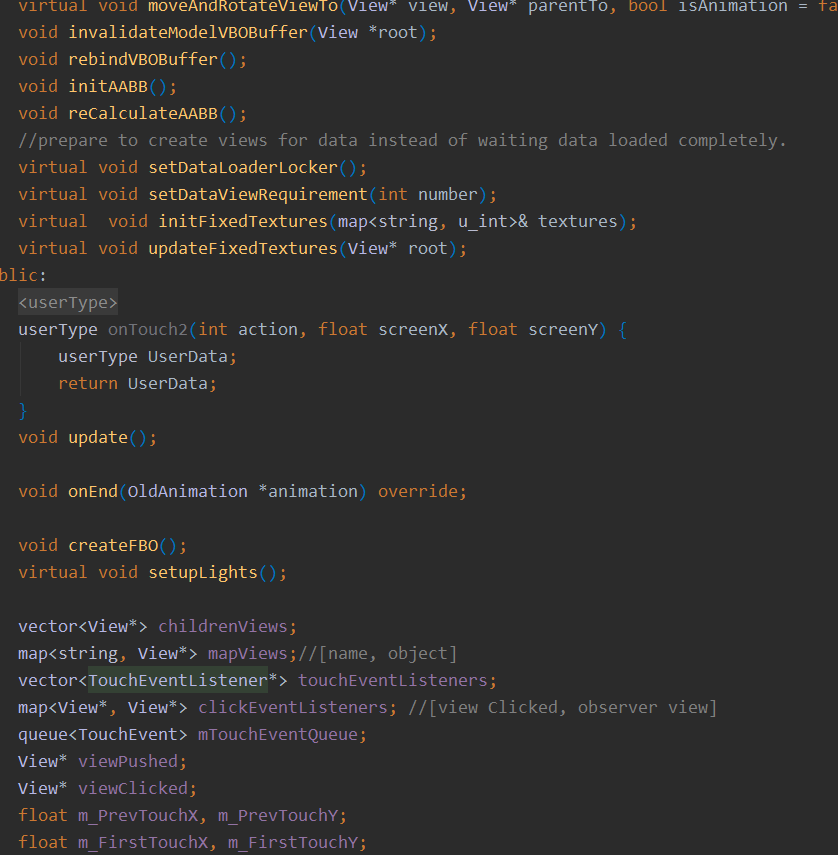
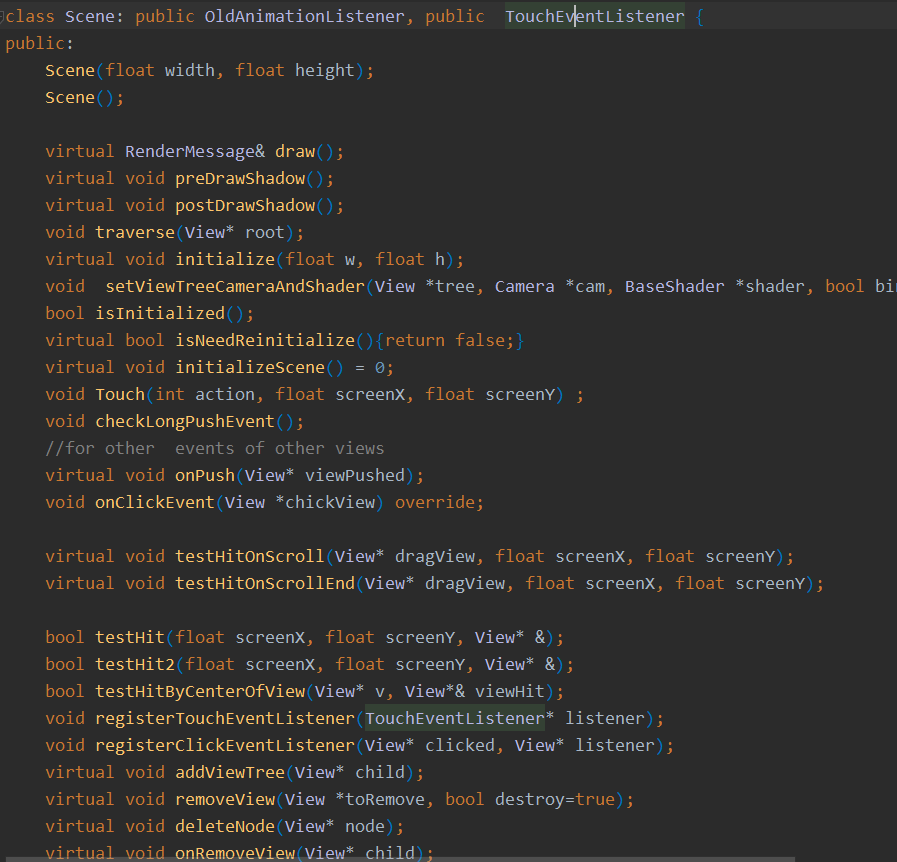
As we can see, all of the elements are 3d model which have massive polygons. The rendering equation which the system used make the whole scene more realistic. The shadow is also real-time rendered by shadow mapping method. The blue sky background is create by skybox method.

The core structure to support such a system consits of two big class. One is called view, the other is the Scene.

A root view class looks like



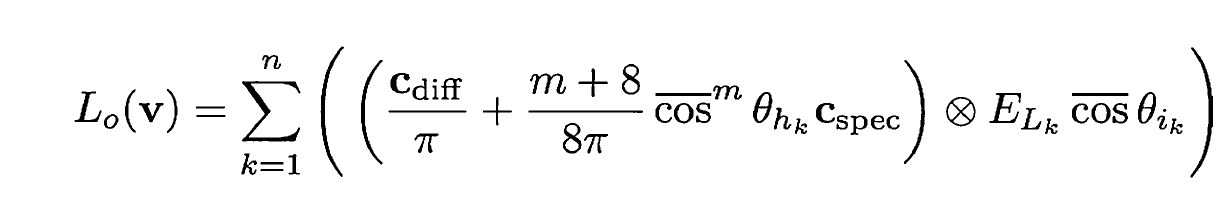
And the class scene looks like this:



The function provided by scene is just like its name indicated. You need a scene to create the whole 3d scene which contains everything you want to create.

**About rendering equations:**

For the reason of being able to run lower-end devices, A simple method was chosen as a standard equation.

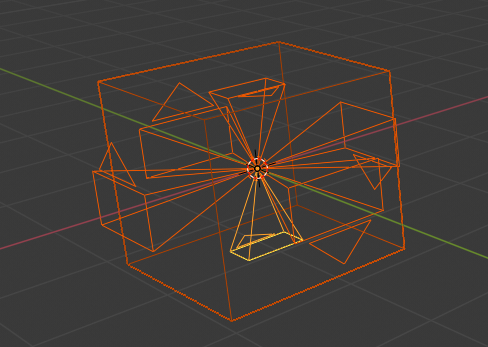


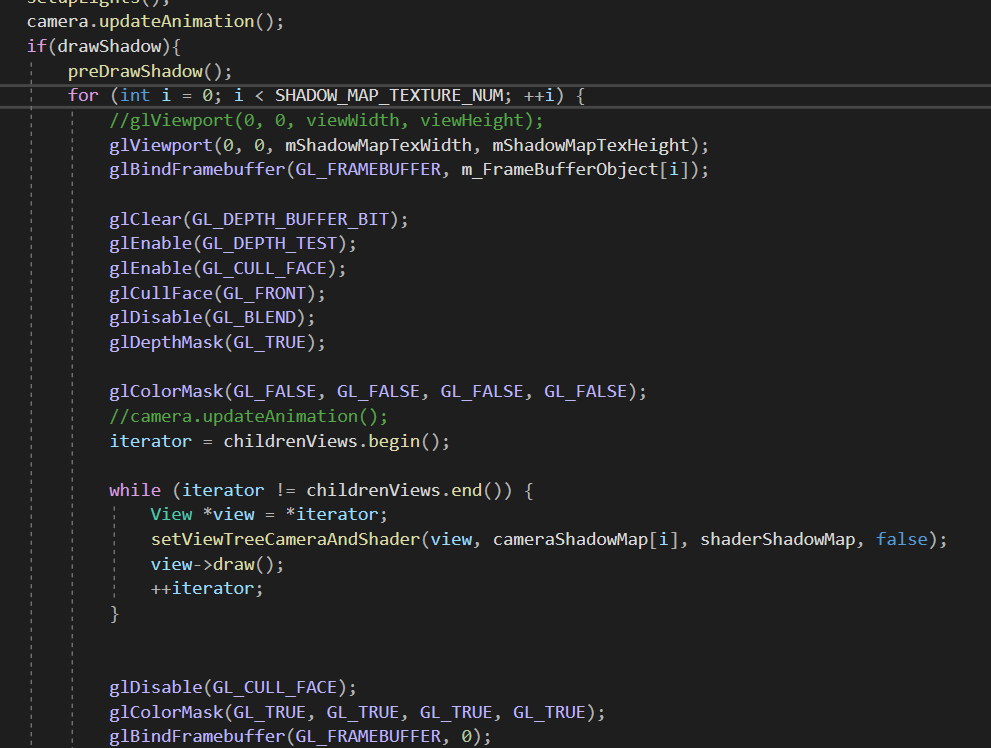
This analytic equation is definitely amazing method which simulated a satisfying light effect with a very cheap cost.

To enhance the realistic sense, Normal map module was also added as an option.

**About shadow mapping**

There are some problems in the development of shadow mapping. One of them is the scene rendered from the light view is different from the one rendered from the view of eye. And that means there are some points whose depth are not determined. So I have found out a solution which is building six cameras. Each of them corresponds to one of the six faces in a cube. And the center of the projection exactly locates at the center of the cube. It means we need 6 passes of rendering for the reason of covering the whole scene. So it’s an expensive solution. However it covered the whole scene which means no matter how is the location of light changed in the world we can easily render the shadow .





**About Collision detection**:

Using Bounding Volume Hierarchies for testing in a broad phase. The following list are the principle to build a BV Tree

1 Each node in the hierarchy should be of minimal volume.

2 The sum of all bounding volumes should be minimal.

3 Greater attention should be paid to nodes near the root of the hierarchy.

4 The volume of overlap of sibling nodes should be minimal

5 The hierarchy should be balanced with respect to both its node structure and its

Content.

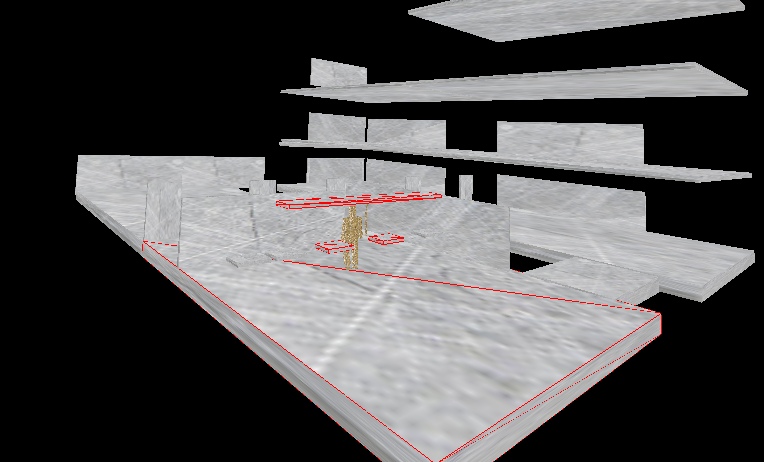
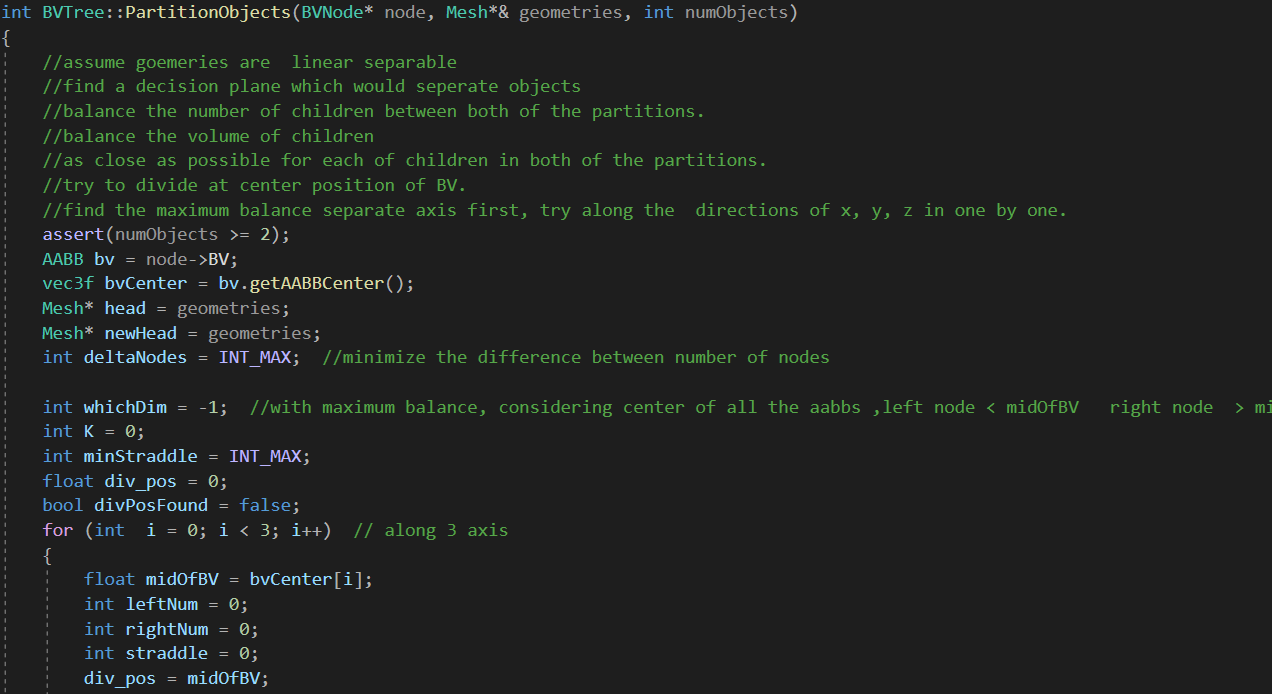


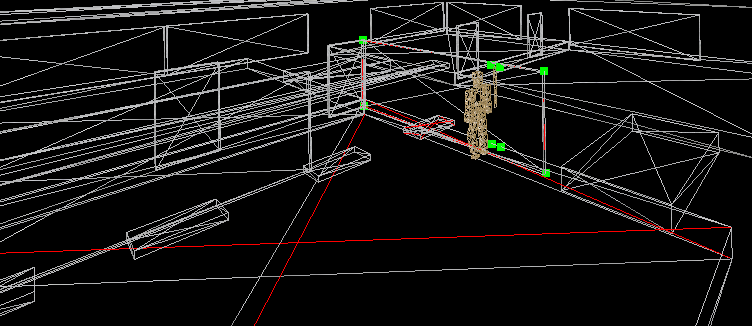
Figure above shows the four of red colorful AABB bounding volumes in the whole scene has been detected. The four AABB will be transported to the next phase which is a narrow phase detection.

As we’ve known , broad phase detection was set up to reduce the cost of the whole costs of detection. One of the key factors about designing a Bounding Volume hierarchy is the decision function which decides how the objects are divided. Using three axis-aligned planes is an option for this kind of decision. Part of code is like this:



**About narrow phase of collision detection**

Here we can see the green points which are the contact points. The points are generated in the narrow phase. Here I’ve used two different methods. One of them is GJK algorithm. The other one is separating axis method which is more easier to find the contact points. We can define these contact points as a manifold with which you can add to next phase to determine the velocities and positions of the collision objects. It’s an important part of whole solver.



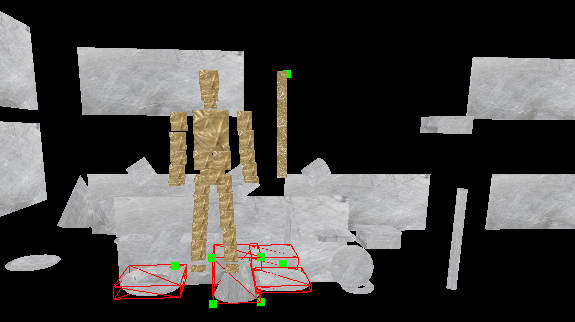


Figure above shows character stand above the 3d objects with gravity is being set. It means collision detection module has extended the domain of the solution for physics simulation.