6.0 SYSTEM IMPLEMENTATION

6.1 Introduction

The implementation phase of the system development is concerned with translating design specifications into a real working version using the chosen programming language which is C++ programming language. This system implementation phase involves the system development environment and program coding. In this phase, the system requirements and design are converted into program code. Each module in this project was developed separately and later integrated into a fully functional system once every module has been tested successfully.

6.2 Development Environment

The choice of development environment is very important because it may affect the outcome of the system development. The correct combinations of hardware and software will ensure that the development process can be performed more effectively and efficiently. However, the wrong choice of combination would be disastrous as the system would not be well developed and impact the quality of the project. The configurations of hardware and software tools used in the development of this system are discussed as below:

6.2.1 Hardware Configuration

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Pentium 4 2 GHz or equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>512MB of RAM (1GB recommended)</td>
</tr>
<tr>
<td>Hard Drive</td>
<td>As little as 50 MB of free space</td>
</tr>
<tr>
<td>Additional Hardware/Devices</td>
<td>Other standard devices such as mouse, keyboard etc.</td>
</tr>
</tbody>
</table>

6.2.2 Software Configuration

Operating System : Window XP or Vista
Framework : OpenGL (already existing in Microsoft Windows XP or Vista)

GLUT library
Programming Language: Visual C++

Additional software: Microsoft Visual Studio 2008

Chosen BV: Oriented Bounding Box (OBB)

Algorithms used:

<table>
<thead>
<tr>
<th>Name:</th>
<th>Description:</th>
</tr>
</thead>
</table>
| Quick Hull algorithm | Tim Lambert at the University of New South Wales
| Creation of the convex hull used for implementing oriented bounding box. |
| Covariance method for triangles | Stefan Gottschalk in his PhD dissertation at University of North Carolina at Chapel Hill
| Creation of the actual oriented bounding boxes. |
| Separating Axis Test for bounding boxes | Christer Ericson in *Real-Time Collision Detection*
| Checking for the intersection of bounding boxes. |
| Faster Triangle-Triangle Intersection Tests | Olivier Devillers and Philippe Guigue in Faster Triangle-Triangle Intersection Tests
| Checking for the intersection of triangles. |
| Checking for the intersection of a ray and an oriented bounding box | Philip J. Schneider and David H. Eberly in Geometric Tools for Computer Graphics
| Select objects on the screen. |
6.3 System Function Code

The codes shown below may not be in their entirety for readability since constraints of available space. Thus, only snippets of interesting code may be shown here.

6.3.1 Import model

```cpp
void ModelManager::refresh()
{
    // SOME CODE HERE
    string name;
    do
    {
        if ((findFileData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) == 0)
        {
            int dot = -1;
            int length = wcslen(findFileData.cFileName);
            name.resize(length);
            for (int i = 0; i < length; i++)
                name[i] = (char)findFileData.cFileName[i];
            if (mOBBsByFile.find(name) == mOBBsByFile.end())
            {
                WavefrontObj* obj = WavefrontObj::open(name.c_str());
                if (obj != NULL)
                {
                    string objName = obj->getName();
                    if (mOBBsByName.find(objName) != mOBBsByName.end())
                    {
                        char buff[33];
                        string newName = objName;
                        sprintf_s(buff, "%d", i);
                        newName.append(buff);
                        for (int i = 3; mOBBsByName.find(newName) != mOBBsByName.end(); i++)
                        {
                            newName = objName;
                            sprintf_s(buff, "%d", i);
                            newName.append(buff);
                        }
                        objName = newName;
                        obj->setName(objName);
                        OBBObj*obbObj = new OBBObj(obj);
                        mObjects.push_back(obbObj);
                        mOBBsByFile.insert(make_pair(name, obbObj));
                        mOBBsByName.insert(make_pair(objName, obbObj));
                    }
                    mOBBsByFile.end();
                }
            }
        }
    } while (FindNextFile(hFind, &findFileData) != 0);
}
```
6.3.2 Draw model

```cpp
void display()
{
    glDepthFunc(GL_LEQUAL);
    if(texturing) glEnable(GL_TEXTURE_2D);
    else glDisable(GL_TEXTURE_2D);

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    theCamera.doGLULookAt(); //
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
    drawAxes();
    pane.draw(flags); //
    glutSwapBuffers();
}
```

6.3.3 Create OBB

```cpp
void OBBObj::getOBB(OBB &obb) const
{
    Matrix44f translation;
    translation.toIdentity();
    float* comps = (float*)&mTranslation;
    for (int i = 0; i < 3; i++)
    {
        translation.set(i, 3, comps[i]);
    }

    Matrix44f trans = translation*mRotation;
    obb.center = (Vector3f)(trans*Vector4f(mOBB.center, 1.0f));
    for (int i = 0; i < 3; i++)
    {
        obb.halfWidths[i] = mOBB.halfWidths[i];
        obb.orientation[i] = ((Vector3f)(trans*Vector4f(mOBB.orientation[i], 0.0f))).normalize();
    }
}
```

6.3.4 Move object

```cpp
void CollisionPane::moveX(float dist)
{
    if (mSelected != NULL)
    {
        mSelected->addTranslation(Vector3f(dist, 0, 0));
        checkCollisions();
    }
}
6.3.5 Orient object

```cpp
void CollisionPane::rotateX(float theta)
{
    if (mSelected != NULL)
    {
        Matrix44f rotation;
        rotation.rotateX(theta);
        mSelected->addRotation(rotation);
        checkCollisions();
    }
}
```

6.3.6 Check OBB for intersections

```cpp
bool Collision::OBB0OBBIntersect(const OBB &a, const OBB &b)
{
    // SOME CODE HERE

    // Test axis L = A0 x B0
    radiusB = b.halfWidths[1] * absRot[0][2] + b.halfWidths[2] * absRot[0][1];
        return false;

    // Test axis L = A0 x B1
    radiusB = b.halfWidths[0] * absRot[0][2] + b.halfWidths[2] * absRot[0][0];
        return false;

    // SOME MORE CODE HERE

    // Since no separating axis found, the OBBs must be intersecting
    return true;
}
6.3.7 Check triangle for intersections

```cpp
bool Collision::TriangleTriangleIntersect(const Triangle3f& t1, const Triangle3f& t2)
{
    float dot11, dot12, dot13;
    dot11 = FourVectorDet(t2.p1, t2.p2, t2.p3, t1.p1);
    dot12 = FourVectorDet(t2.p1, t2.p2, t2.p3, t1.p2);
    dot13 = FourVectorDet(t2.p1, t2.p2, t2.p3, t1.p3);

    // If the dots are all zero then the two triangles are
    // coplanar.
    if (IsZero(dot11) && IsZero(dot12) && IsZero(dot13))
    {
        return PointInTriangle(t2.p1, t1) || PointInTriangle(t2.p2, t1) ||
            PointInTriangle(t2.p3, t1) ||
            PointInTriangle(t1.p1, t2) ||
            PointInTriangle(t1.p2, t2) ||
            PointInTriangle(t1.p3, t2);
    }
    else if ((dot11 > 0 && dot12 > 0 && dot13 > 0) ||
              (dot11 < 0 && dot12 < 0 && dot13 < 0))
    {
        return false;
    }
    float dot21 = FourVectorDet(t1.p1, t1.p2, t1.p3, t2.p1);
    float dot22 = FourVectorDet(t1.p1, t1.p2, t1.p3, t2.p2);
    float dot23 = FourVectorDet(t1.p1, t1.p2, t1.p3, t2.p3);

    // If All dots are non zero and of the same sign,
    // then the vertices of t1 are all on one side of the plane
    // of t2. Therefore, the triangles cannot be intersecting.
    if ((dot21 > 0 && dot22 > 0 && dot23 > 0) ||
        (dot21 < 0 && dot22 < 0 && dot23 < 0))
    {
        return false;
    }
    Triangle3f newT1 = t1, newT2 = t2;
    bool doT2Swap = CircularPermutation(newT1.p1, newT1.p2, newT1.p3,
                                          newT2.p1, newT2.p2, newT2.p3,
                                          dot11, dot12, dot13);
    bool doT1Swap = CircularPermutation(newT2.p1, newT2.p2, newT2.p3,
                                          newT1.p1, newT1.p2, newT1.p3,
                                          dot21, dot22, dot23);

    if (doT1Swap) swap(newT1.p2, newT1.p3);
    if (doT2Swap) swap(newT2.p2, newT2.p3);

    float determ = FourVectorDet(newT1.p1, newT1.p2, newT2.p1, newT2.p2);
    if (IsZero(determ) || determ < 0)
    {
        determ = FourVectorDet(newT1.p1, newT1.p3, newT2.p2, newT2.p3);
        if (IsZero(determ) || determ < 0)
            return true;
    }
    // if (IsZero(determ) || determ < 0)
    return false;
}
```

6.3.8 Highlight intersections

```cpp
void OBBObj::setBoxColor(float r, float g, float b)
{
    mBoxColor[0] = r;
    mBoxColor[1] = g;
    mBoxColor[2] = b;
}
```

6.3.9 Export data

```cpp
if (!intersect)
{
    sequence += 1;
    cout << "Motion #" << sequence << endl;
    cout << "COLLIDING:\tNO" << endl;
    cout << "\tBOX 1:" << endl;
    cout << "\tX: " << box1.center.x << endl;
    cout << "\tY: " << box1.center.y << endl;
    cout << "\tZ: " << box1.center.z << endl;
    cout << "\tBOX 2:" << endl;
    cout << "\tX: " << box2.center.x << endl;
    cout << "\tY: " << box2.center.y << endl;
    cout << "\tZ: " << box2.center.z << endl;
    sequence += 1;
    logFile << "Motion #" << sequence << endl;
    logFile << "COLLIDING:\tNO" << endl;
    logFile << "\tBOX 1:" << endl;
    logFile << "\tX: " << box1.center.x << endl;
    logFile << "\tY: " << box1.center.y << endl;
    logFile << "\tZ: " << box1.center.z << endl;
    logFile << "\tBOX 2:" << endl;
    logFile << "\tX: " << box2.center.x << endl;
    logFile << "\tY: " << box2.center.y << endl;
    logFile << "\tZ: " << box2.center.z << endl;
    continue;
}
```