

بسم الله الرحمن الرحيم و الصلاة و السلام على نبينا محمد صلى الله عليه و سلم

مقدمة :

هناك العديد من مكتبات التي تتخصص في كشف التصادم و الديناميك الحرة منها و التجارية , و تختلف أهدافها حسب أهداف المطور. و من أهم مكتبات كشف التصادم و الديناميك ODE و BULLET . تتفوق BULLET على ODE في المعالجة المتوازية . و تدعم BULLET أقوى المنصات linux و mac و windows و playstation 3 . Bullet هي مكتبة موجهة للتصادمات و الديناميك و تم تصميمها بلغة ++C . و هي مكتبة مفتوحة المصدر و مفتوحة على الاستعمال التجاري بترخيص zlib .

كيفية التثبيت :

حمل الملف المصدري لآخر إصدار للمكتبة من Google Code
<http://code.google.com/p/bullet/downloads/list>

• على ويندوز:

أدخل على المجلد msvc و من ثم src و من ثم افتح المشروع BULLET_PHYSICS.sln ببرنامج Microsoft Visual Studio و قم بترجمته .

• على لينكس أو ماك :

نستعمل أحد البرامج التالية MSVC , KDevelop , Xcode أو أي برنامج للترجمة أو من خلال الطرفية.

عموما يفضل تثبيت المكتبة من السنابتيك على أوبنتو أو يوميكس على فيدورا . و يتوفر الملف المصدري Bullet على أمثلة كثيرة و جيدة باستخدام Glut , أنظر إلى المجلد Demos .

راجع ملف التثبيت المرفق مع مكتبة Bullet.

أولا و كل شيء , كيف تتم آلية بناء برنامج ب Bullet ؟

- 1 : إعداد عالم الديناميك .
- 2 : إعداد الأجسام و ربطها بعالم الديناميك.
- 3 : إعداد كتلة كل جسم , الكتل الموجبة تمثل الأجسام المحررة , و الصفرية للأجسام الثابتة .
- 4 : إنشاء المحنط Triangle Mesh , Convex Hull , Sphere , Cone , Box .
- 5 : إعداد الخصائص الفيزيائية مثل الاحتكاك .
- 6 : إعداد دالة التحديث stepSimulation .

توفر مكتبة Bullet مجموعة من المتغيرات تجعلك في غنى عن إنشاءها.

btScalar : و هو متغير إما أن يكون float أو double .

لاستخدام المتغيرات من نوع double نقوم بالتعريف الدابل أولا.

```
define BT_USE_DOUBLE_PRECISION
```

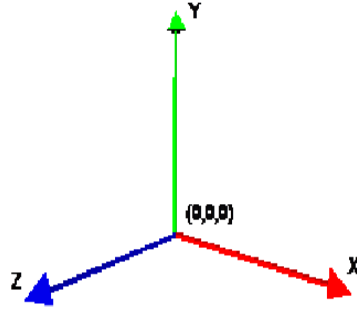
أما في حالة استخدام متغيرات float فنستخدم BtScalar مباشرة.

BtVector3 : تخص كل المتجهات و الأشعة و كل ما له ثلاثة قيم.

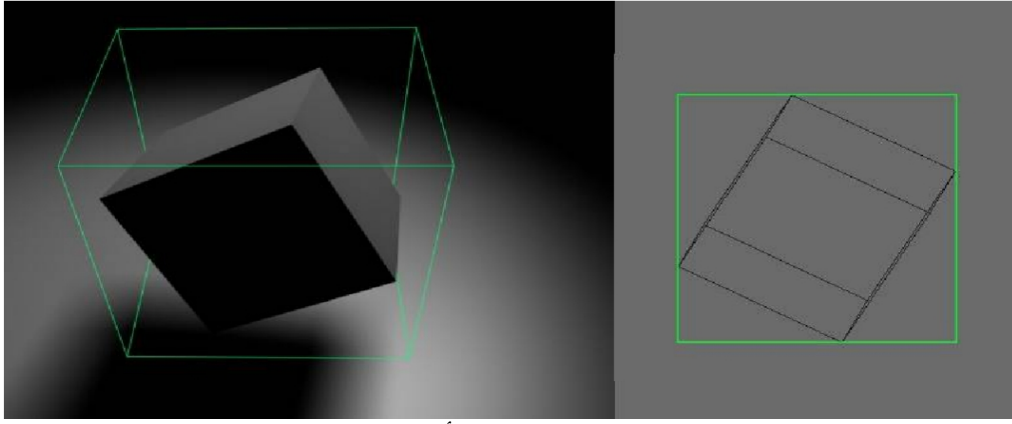
btQuaternion و btMatrix3x3 : تخص الدوران.

BtTransform : يخص التحويل و الدوران.

و هذا هو نظام التحويل في Bullet :



btTransformUtil و btAabbUtil: يوفران دوال التحويل و **aabb**.
(Aligned Axis Bounding Box) أو **aabb**: وهو صندوق يحوي بداخله الجسم سيارة أو كرة أو صندوق مثلا , و يأخذ الاتجاه **X** و **y** و **z** دائما و لا يدور حول مركزه و إنما يتحرك فقط . و هذا لزيادة كفاءة كشف التصادم . سمه الصندوق المحنط الثابت التوجيه.

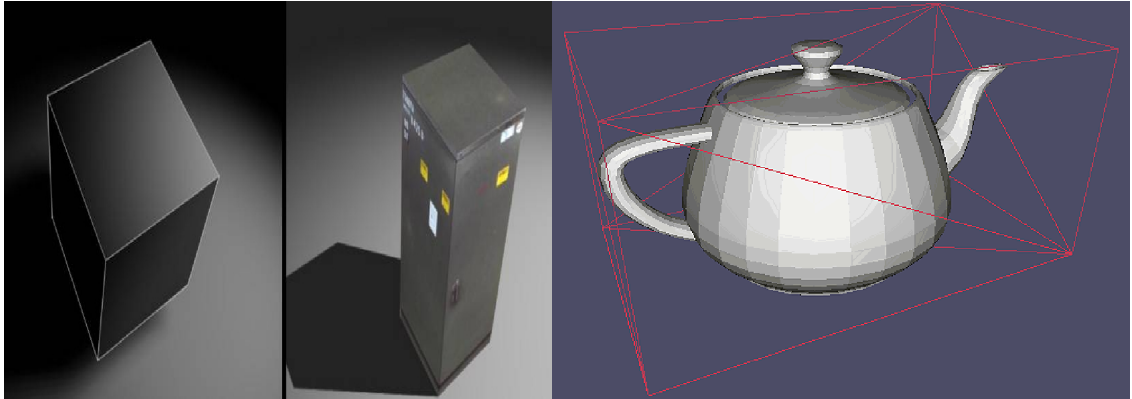


الصورة تمثل المحنط باللون الأخضر و الصندوق .

ماذا يفيد **aabb** بالضبط ؟

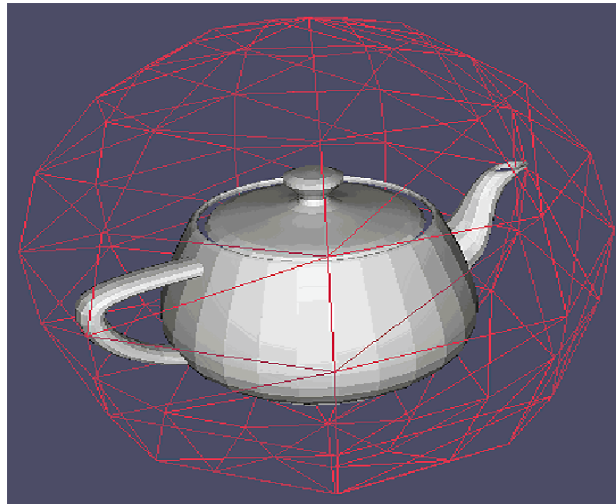
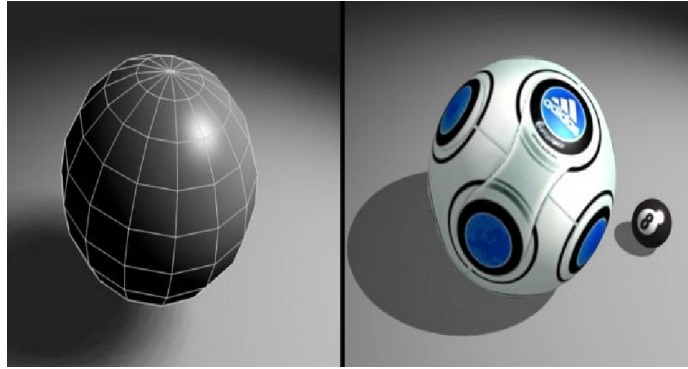
لنفرض أنه لدينا سيارة و طائرة و أشجار و و و و كلها عبارة عن مجموعة شبكيات هائلة , فكشف التصادم عندها سيكون عويصا جدا . لأن الأصل في **Bullet** المقارنة بين مثلثات الشبكيات . و لكن هناك طريقة مثلى و هي كشف التصادم بين الصناديق أولا . فكل مجسم نضعه في صندوق , السيارة في صندوق و الطائرة في صندوق و و و و . و من ثم نكشف التصادم بين الصناديق و عندما يتصادم صندوق السيارة مع صندوق الطائرة نقوم بكشف التصادم الحقيقي بين السيارة و الطائرة , أضناك فهمت ما أرمي إليه .

أدوات التحنيط: و غرضها زيادة كفاءة **Bullet** في سرعة كشف التصادم و الديناميك .
btBoxShape: المحنط الصندوق (ليس **aabb**) .

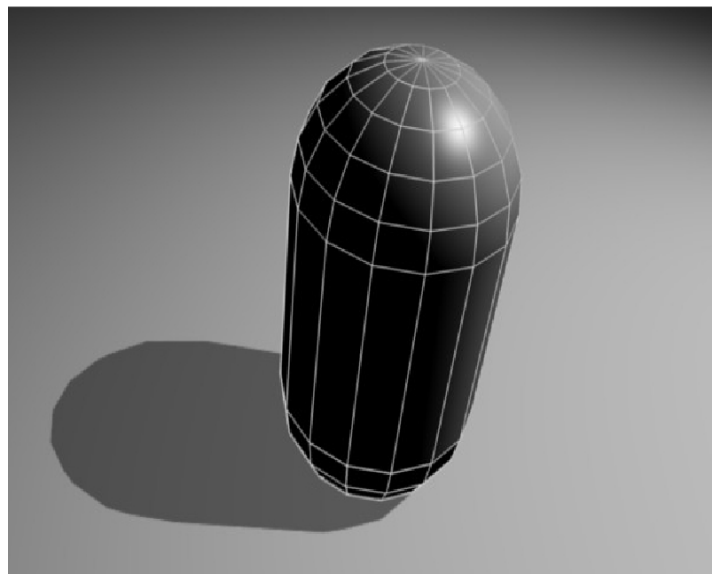


هذا النوع من التحنيط لديه دوران و تحريك فقط .
btSphereShape:

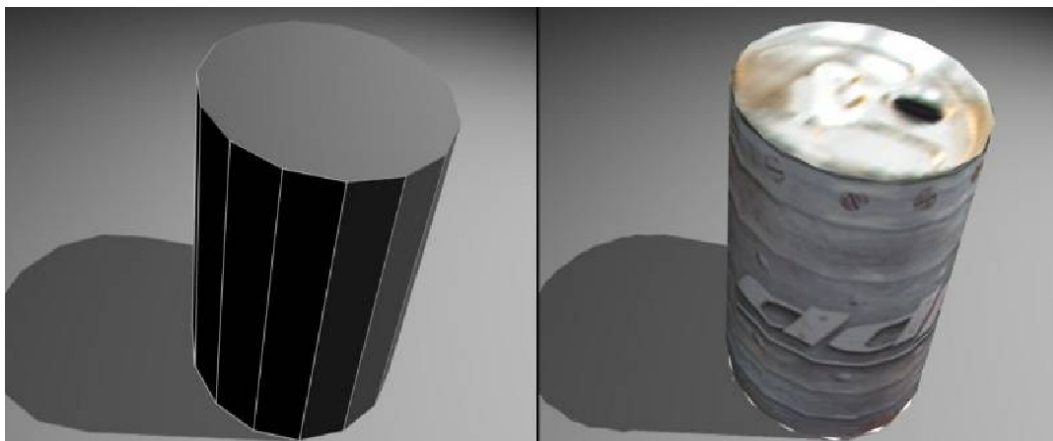
التحنيط الكرة.



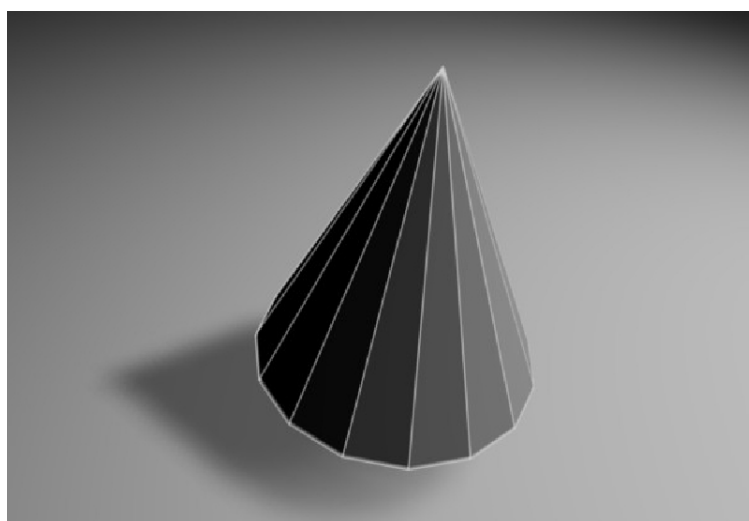
:btCapsuleShape



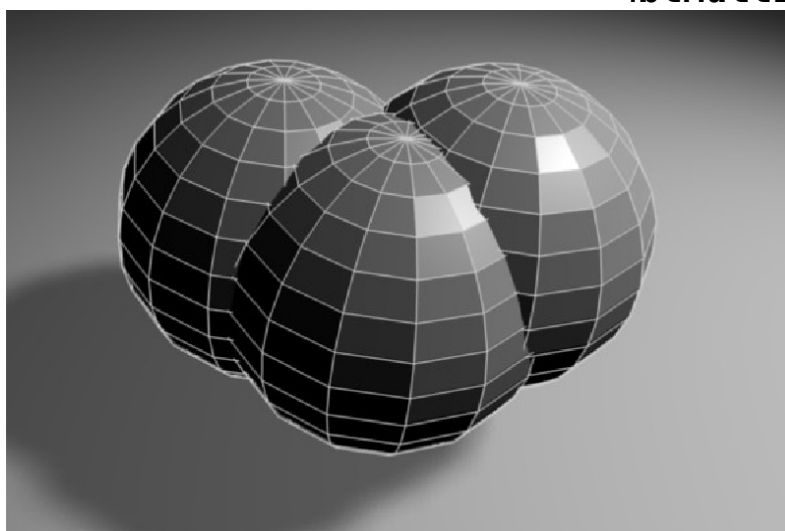
:btCylinderShape



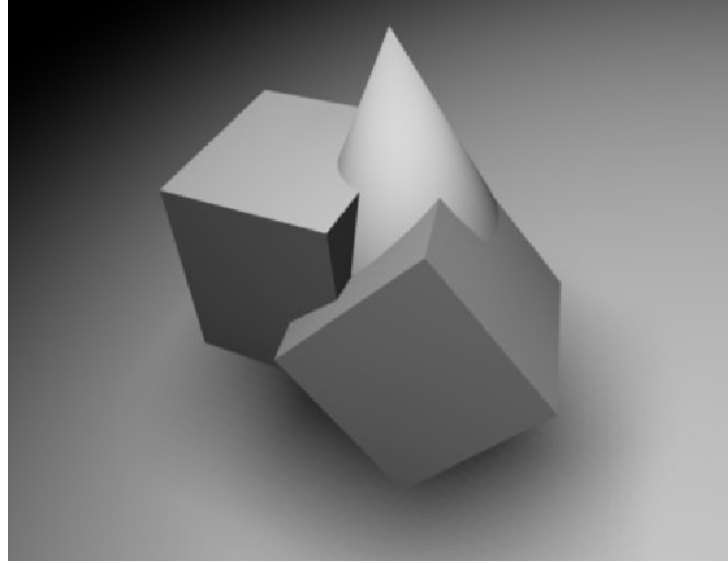
: **btConeShape**



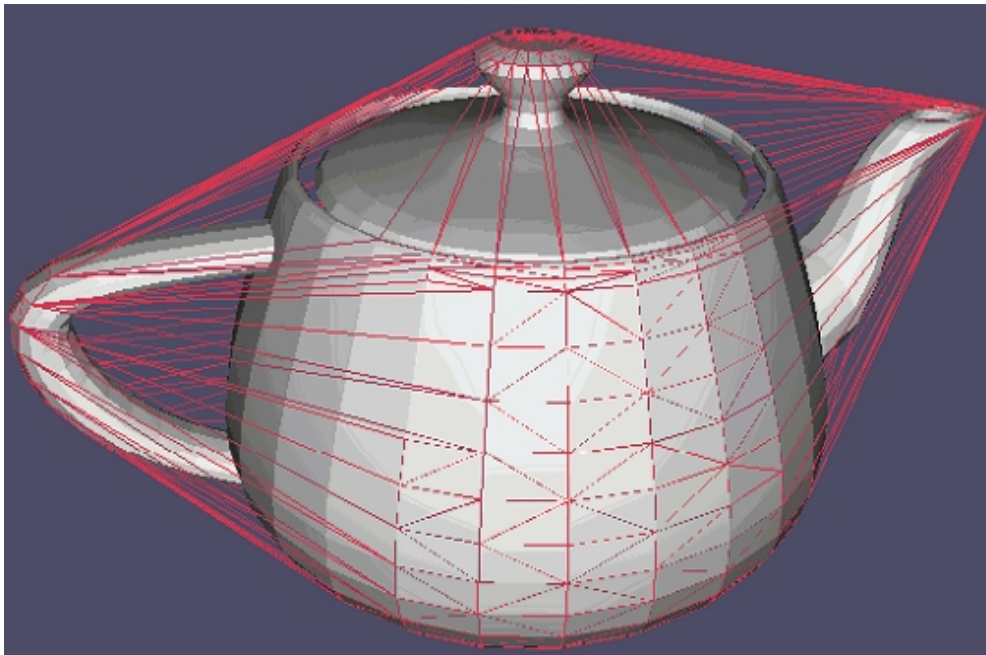
: **btMultiSphereShape**



btCompoundShape : تركيب مجموعة من المحنطات في محنط واحد.

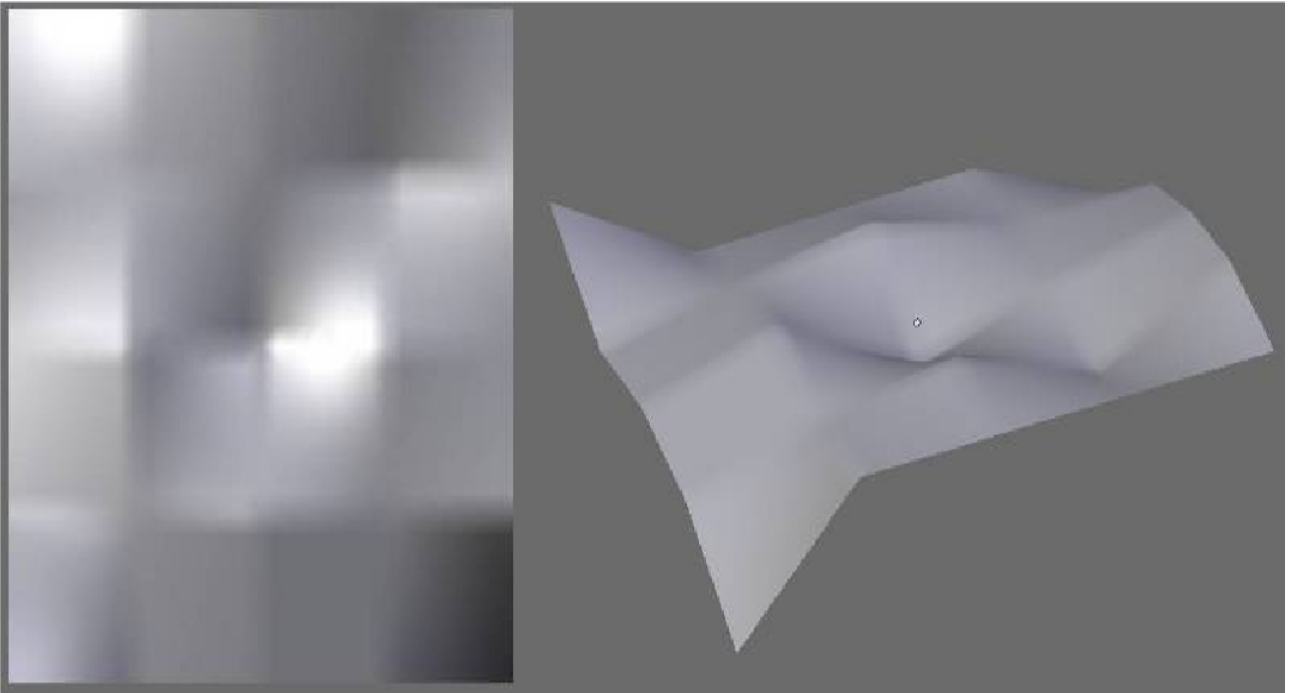


btConvexHullShape :
أنظر إلى المحنط هول ممثل بالخطوط الحمراء.



BtBvhTriangleMeshShape :
هذا النوع من التحنيط يستعمل في الأجسام الساكنة التي لا تتحرك أبدا كالأرضية و الجدران ووو .

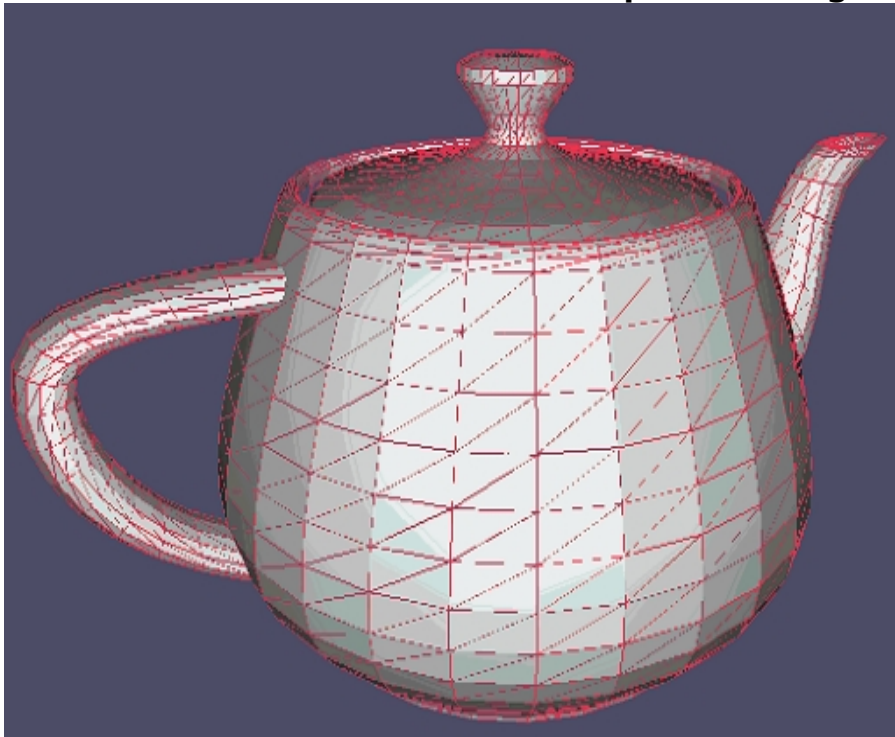
BtHeightfieldTerrainShape



: btStaticPlaneShape

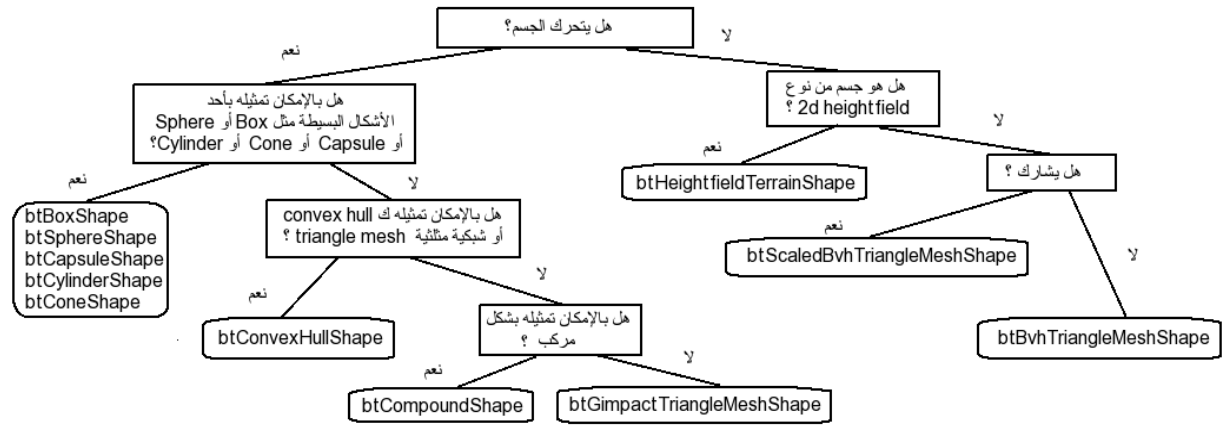
هذا النوع من التحنيط يظهر من اسمه.

: BtGimpactTriangleMeshShape



هذا النوع من التحنيط يشمل جميع الأشكال التي تتحرك .

هذا المخطط يمثل توجيهه لإنشاء التصادم:



هذا الجدول مهم جدا و يعطي أفضل النتائج فيعطيك القرارات التي ستستخدمها في إنشاء التصادمات.

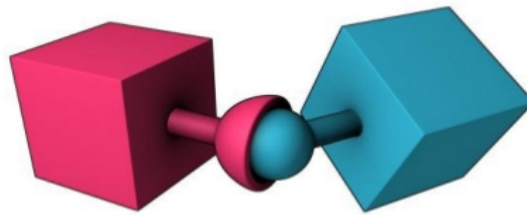
ما هو Gimpact؟

و هي أيضا مكتبة كشف التصادمات, تم إدماجها مع Bullet , تعمل على كشف التصادمات بين الشبكات المثلثية المتحركة والأجسام التي لا يمكن تمثيلها بأحد المحنطات أو تركيبها.

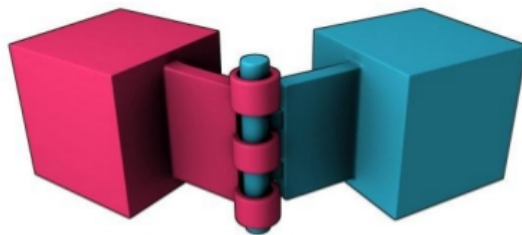
المقيدات Constraints :

توفر مكتبة Bullet العديد من التقيدات , و يتطلب إنشاءها جسمين على أقل أحدها متحرك أو حر, و هو قد تكون التقيدات في الغالب أزرار أو فواصل أو أو أو .

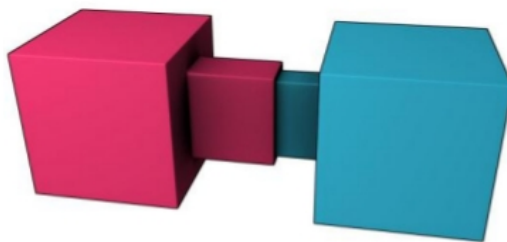
تقييد نقطة لنقطة : btPoint2PointConstraint .



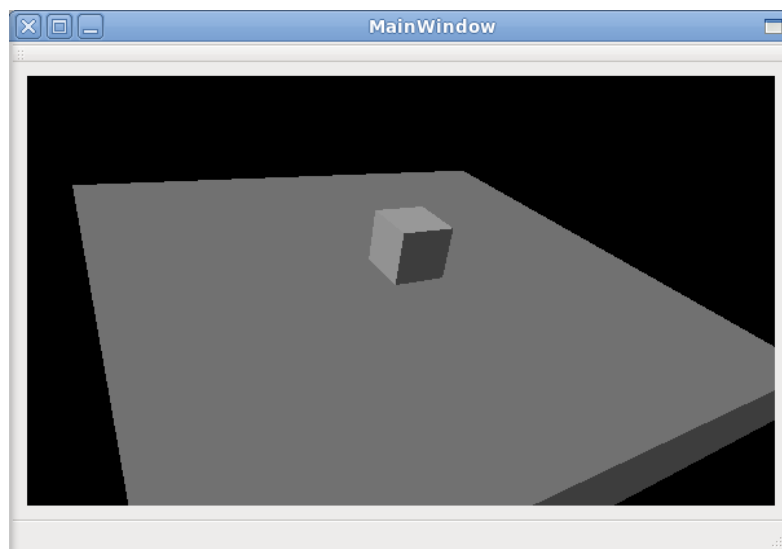
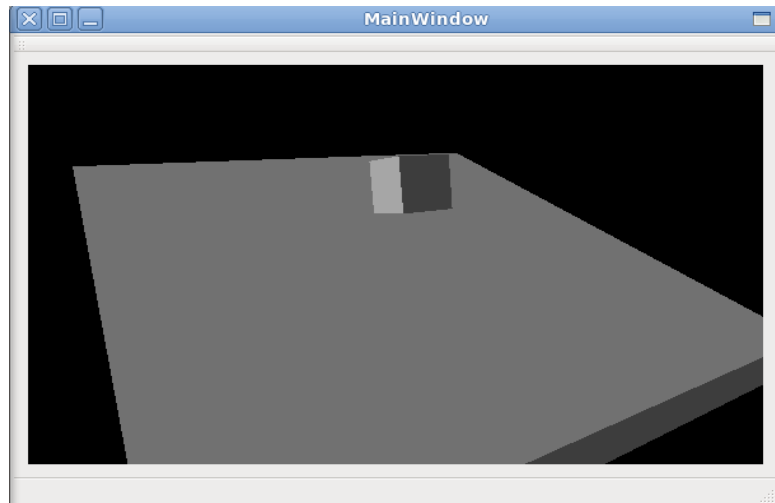
تقييد المفصل : btHingeConstraint .



تقييد مزلاج : btSliderConstraint .



مثال :



سنقوم بعمل محاكات لصندوق يسقط على الأرضية ثم يرتد لغاية أن يستقر و ذلك باستعمال مكتبة Bullet, المثال بسيط جدا و هو يمثل أحد أسس البرمجة بمكتبة بوليت. لنقوم بإنشاء فئة ما ولنسميها مثلا GLBullet .

```
class GLBullet
{
public:

    GLBullet();
    void Init();
    void Sim();
    void end();
    void cube();
    void sol();

    btDiscreteDynamicsWorld* myWorld;
    btTypedConstraint *mytc;
    btBroadphaseInterface* myBroadphase;
    btCollisionDispatcher* myDispatcher;
    btDefaultCollisionConfiguration* myCollisionConfiguration;
    btGImpactMeshShape *H;
```

```

btSequentialImpulseConstraintSolver *mySequentialImpulseConstraintSolver;

btDefaultMotionState *myMotionState, *myMotionState_Sol;

btScalar matrix[16];
};

```

لدينا الدالة و هي تمثل الأساس في البرنامج, يتم استدعاءها مرة واحدة في البرنامج.
 باقي الدوال الأخرى سنتطرق إليها لاحقاً.

```
BtDiscreteDynamicsWorld* myWorld;
```

المتغير btDiscreteDynamicsWorld يقوم بإنشاء العالم الذي سيجري فيه حساب كشف التصادمات و الديناميك.

```
BtBroadphaseInterface* myBroadphase;
```

الفئة btBroadphaseInterface تقوم بعمل وضع معلمات لزيادة سرعة كشف التصادم .

```
BtCollisionDispatcher* myDispatcher;
```

الفئة BtCollisionDispatcher تقوم بإعداد الخوارزمية المناسبة لمعالجة التصادم.

```
btDefaultCollisionConfiguration* myCollisionConfiguration;
```

الفئة btDefaultCollisionConfiguration تقوم بالإعدادات الخاصة ببوليت و تشمل الذاكرة فقط.

```
btGImpactMeshShape *H;
```

الفئة btGImpactMeshShape تقوم بإدارة ومعالجة الشبكيات المثلثية, يتم إضافتها غالباً (لا أعرف السبب بالضبط).

```
btSequentialImpulseConstraintSolver *mySequentialImpulseConstraintSolver;
```

الفئة btSequentialImpulseConstraintSolver تمثل تطبيق خاص بال Projected Gauss Seidel .

```
btDefaultMotionState *myMotionState, *myMotionState_Sol;
```

الفئة btDefaultMotionState عبارة عن تطبيق أو أسلوب يقوم بإدارة التحولات في عالم البوليت.

```
btScalar matrix[16];
```

و أخيراً matrix و نستخدمها في التحريك.

و الآن إلى الدالة Init و هي على التوالي :

```

void glbullet::Init(){
    myCollisionConfiguration = new btDefaultCollisionConfiguration();
    myDispatcher = new btCollisionDispatcher(myCollisionConfiguration);
    myBroadphase = new btDbvtBroadphase();
    mySequentialImpulseConstraintSolver = new
btSequentialImpulseConstraintSolver;
    myWorld = new
btDiscreteDynamicsWorld(myDispatcher,myBroadphase,mySequentialImpulseConstraint
Solver,myCollisionConfiguration);
    myWorld->setGravity( btVector3(0,-10,0) );
    cube();
    sol();
}

```

نمرر المواصفات التي أنشئت لعالم البوليت في الدالة btDiscreteDynamicsWorld .

```
myWorld->setGravity( btVector3(0,-10,0) );
```

هذا السطر يقوم بإعداد جاذبية عالم البوليت. و من ثم نقوم باستدعاء دالتا cube و sol.

```

void glbullet::cube(){
    ////////////////////////////////////////////////// box ///////////////////////////////////
    btCollisionShape* shape = new btBoxShape( btVector3(1,1,1) );

    btTransform myTransform;
    myTransform.setIdentity();
    myTransform.setOrigin( btVector3(0,5,0) );
    myTransform.setRotation( btQuaternion(-3.14159 / 2, 0, 0, 1));

    btVector3 localInertia(0,0,0);
    btScalar mass = 0.001f;
    if ( mass )
        shape->calculateLocalInertia( mass, localInertia );
    btRigidBody *body;
    //using motionstate is recommended, it provides interpolation capabilities,
    and only synchronizes 'active' objects
    myMotionState = new btDefaultMotionState(myTransform);
    btRigidBody::btRigidBodyConstructionInfo
myBoxRigidBodyConstructionInfo( mass, myMotionState, shape, localInertia );
    body = new btRigidBody(myBoxRigidBodyConstructionInfo);

    //add the body to the dynamics world
    myWorld->addRigidBody(body);
}

```

```
btCollisionShape* shape = new btBoxShape( btVector3(1,1,1) );
```

هذا السطر يقوم بإعداد كشف لتصادم لصندوق و 1,1,1 تمثل أبعاد الصندوق ابتداء من مركزه.

```

    btTransform myTransform;
    myTransform.setIdentity();
    myTransform.setOrigin( btVector3(0,5,0) );
    myTransform.setRotation( btQuaternion(-3.14159 / 2, 0, 0, 1));

```

يقوم هذا السطر البرمجي بإعداد تموضع.

```

btVector3 localInertia(0,0,0 );
btScalar mass = 0.001f;
if ( mass )
    shape->calculateLocalInertia( mass, localInertia );

```

إعداد الكتلة و مركز العطالة .

```
btRigidBody *body;
```

السطر الأهم و هو أحد مفاهيم التصادمات, إعداد جسم صلب body .

```

//using motionstate is recommended, it provides interpolation
capabilities, and only synchronizes 'active' objects
myMotionState = new btDefaultMotionState(myTransform);
btRigidBody::btRigidBodyConstructionInfo
myBoxRigidBodyConstructionInfo( mass, myMotionState, shape,
localInertia );
body = new btRigidBody(myBoxRigidBodyConstructionInfo);

```

هنا نقوم ببناء الجسم الصلب بودي , و هو يأخذ البرامترات الكتلة و التحريك و كاشف التصادم و مركز العطالة.

```
myWorld->addRigidBody(body);
```

ربط البودي بعالم البولييت.

```

void glbullet::sol(){
    ////////////////////////////////////////////////// SOL ///////////////////////////////////
    btCollisionShape* shape_sol = new btBoxShape( btVector3(10,1,10) );

```

```

//m_collisionShapes.push_back(shape_sol);
btTransform myTransform;
myTransform.setIdentity();
myTransform.setOrigin( btVector3(0,0,0) );
btVector3 localInertiaSol(0,0,0);

btScalar mass = 0;
btRigidBody *body_sol;
// Using motionstate is recommended, it provides interpolation capabilities,
and only synchronizes 'active' objects
myMotionState_Sol = new btDefaultMotionState(myTransform);
btRigidBody::btRigidBodyConstructionInfo rbInfo_sol( mass, myMotionState_Sol,
shape_sol, localInertiaSol );
body_sol = new btRigidBody(rbInfo_sol);

// Add the body to the dynamics world
myWorld->addRigidBody(body_sol);
}

```

نقوم بإنشاء الأرضية بنفس المراحل كالصندوق غير أنه الأرضية تأخذ الكتلة 0 للدلالة على أنها لا تتحرك.

```

void glbullet::Sim(){
    if (myWorld)
    {
        myWorld->stepSimulation( 0.1 );
        myWorld->debugDrawWorld();
    }
}

```

هذه الدالة Sim تقوم بإعادة روتين التصادم و الديناميك و كل ما هو معطى في الدالة Init .
الدالة debugDrawWorld تقوم بإعادة ضبط الحسابات لأجل عرض نتائج جيدة.

```

void glbullet::end(){
    int i;
    for (i=myWorld->getNumCollisionObjects()-1; i>=0 ;i--){
        btCollisionObject* obj = myWorld->getCollisionObjectArray()[i];
        btRigidBody* body = btRigidBody::upcast(obj);
        if (body && body->getMotionState())
        {
            delete body->getMotionState();
        }
        myWorld->removeCollisionObject( obj );
        delete obj;
    }

    //delete dynamics world
    delete myWorld;
    //delete solver
    delete mySequentialImpulseConstraintSolver;
    //delete broadphase
    delete myBroadphase;
    //delete dispatcher
    delete myDispatcher;
    delete myCollisionConfiguration;
}

```

دالة end تمثل مهدم لعالم البوليت و تحرير البيانات من الذاكرة. يتطلب تحرير البيانات بعكس إنشاءها (يجب مراعاة الترتيب).

في فئة التصيير نقوم باستدعاء الدالة `.Init`.

```
void glwidget::initializeGL()
{
    glShadeModel(GL_SMOOTH);           // Enable Smooth Shading
    glEnable(GL_NORMALIZE);
    glClearColor(0.0f, 0.0f, 0.0f, 0.5f); // Black Background
    glClearDepth(1.0f);                 // Depth Buffer Setup
    glEnable(GL_DEPTH_TEST);            // Enables Depth Testing
    glDepthFunc(GL_LEQUAL);             // The Type Of Depth
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    glEnable(GL_LIGHT0);                // Quick And Dirty
    glEnable(GL_LIGHTING);              // Enable Lighting
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST); // Really Nice

    glb.Init(); // <<<-----
}

void glwidget::resizeGL(int width, int height){
    glViewport(0, 0, (GLsizei) width, (GLsizei) height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, (GLfloat) width/(GLfloat) height, 0.1, 1000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.,0.,-20);
    glRotatef(45,1.,1.,0.);
    //glRotatef(15,0.,1.,0.);
}
```

```
void glwidget::paintGL()
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glb.Sim();
    Box(10,1,10);
    glPushMatrix();
    glb.myMotionState->m_graphicsWorldTrans.getOpenGLMatrix( matrix );
    glMultMatrixf( matrix );
    Box(1.,1.,1.);
    glPopMatrix();
    swapBuffers();
}
```

استدعاء دالة `Sim` يعمل على تحديث التصادم و الديناميك لكل إطار `Frame`, ثم نقوم بإنشاء أرضية `Box` `10,1,10`, ثم نستخرج مصفوفة التحويل من التحويل `myMotionState`. ثم نمررها إلى الدالة `.glMultMatrixf`.

و أخيرا نستدعي دالة هدم `Bullet` في المهدم `glwidget`.

```
glwidget::~~glwidget(){
    glb.end();
}
```

```
void glwidget::Box(float x, float y, float z)
{
    glPushMatrix();
    glScalef(x,y,z);
    glBegin (GL_QUADS);
    //face 1
```

```

glNormal3i(-1, 1,-1);
glVertex3i(-1, 1,-1); glVertex3i( 1, 1,-1);
glVertex3i( 1,-1,-1); glVertex3i(-1,-1,-1);
//face 2
glNormal3i(-1,-1,-1);
glVertex3i(-1,-1,-1); glVertex3i( 1,-1,-1);
glVertex3i( 1,-1, 1); glVertex3i(-1,-1, 1);
// face 3
glNormal3i( 1,-1, 1);
glVertex3i( 1,-1, 1); glVertex3i( 1,-1,-1);
glVertex3i( 1, 1,-1); glVertex3i( 1, 1, 1);
//face 4
glNormal3i( 1, 1,-1);
glVertex3i( 1, 1,-1); glVertex3i(-1, 1,-1);
glVertex3i(-1, 1, 1); glVertex3i( 1, 1, 1);
//face 5
glNormal3i(-1, 1, 1);
glVertex3i(-1, 1, 1); glVertex3i(-1, 1,-1);
glVertex3i(-1,-1,-1); glVertex3i(-1,-1, 1);
//face 6
glNormal3i( 1,-1, 1);
glVertex3i( 1,-1, 1); glVertex3i( 1, 1, 1);
glVertex3i(-1, 1, 1); glVertex3i(-1,-1, 1);
glEnd();
glPopMatrix();
}

```

أرجوا التوفيق من الله