

情報メディア実験A

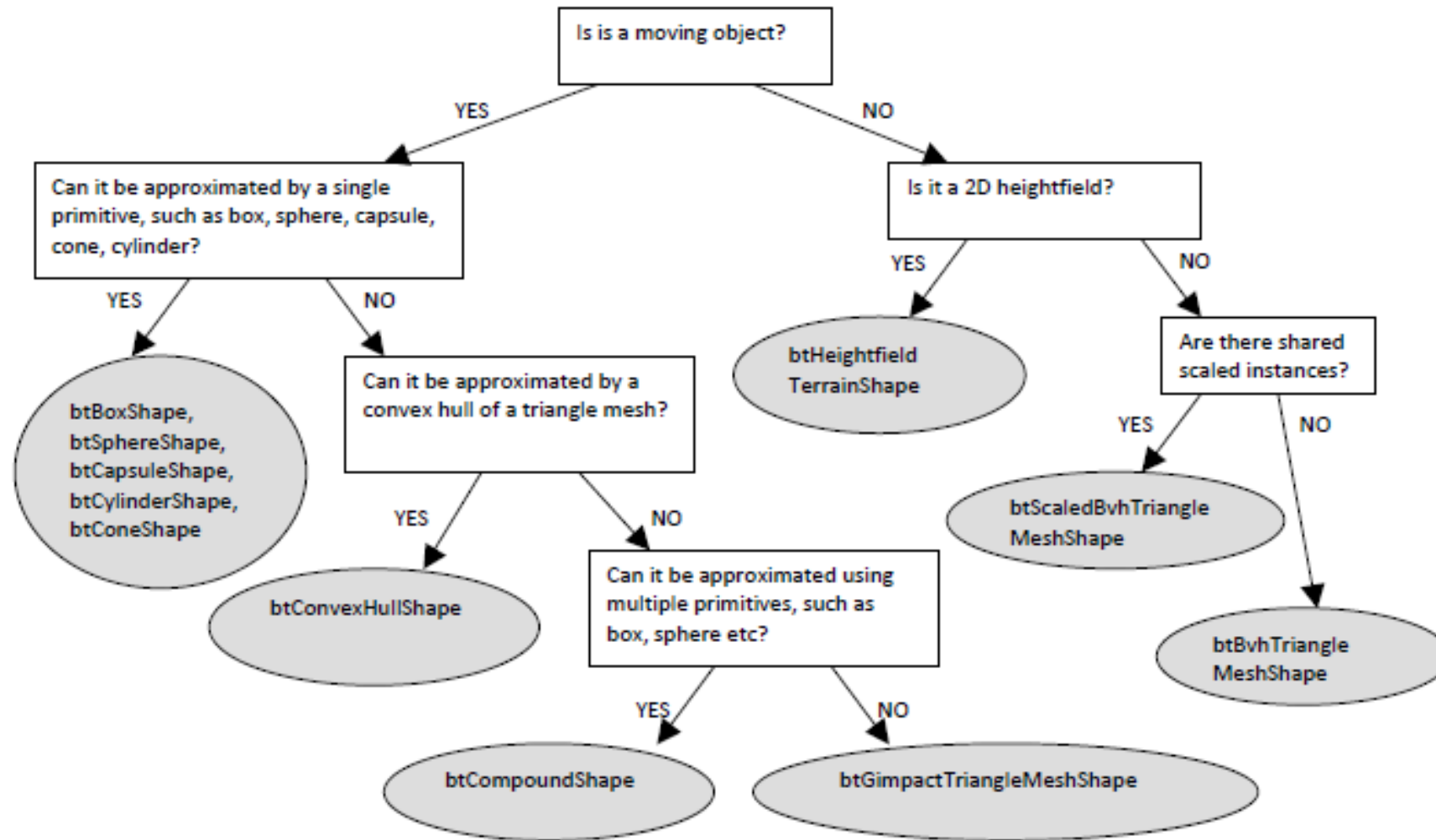
# 物理エンジンを使った アプリケーション開発

筑波大学情報学群  
情報メディア創成学類  
藤澤誠

# 任意形状の読み込み

- Bulletで扱える形状 (bt\*Shape)
  - プリミティブ(基本形状) :  
Box, Sphere, Capsule, Cylinder, Cone,  
(MultiSphere:capsule,convex hull用)
  - 複合形状 : Compound
  - 無限遠平面 : StaticPlane
  - ハイトフィールド : HeightfieldTerrain
  - 凸包 : ConvexHull -> 凹んだところがない多角形  
(形状は三角形ポリゴンで表される)
  - 三角形ポリゴン :  
BvhTriangleMesh (static object用)  
GImpactTriangleMesh

# 任意形状の読み込み



# 任意形状の読み込み

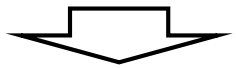
- Bulletで扱える形状 (bt\*Shape)
  - プリミティブ(基本形状) :  
Box, Sphere, Capsule, Cylinder, Cone,  
(MultiSphere:capsule,convex hull用)
  - 複合形状 : Compound
  - 無限遠平面 : StaticPlane
  - ハイトフィールド : HeightfieldTerrain
  - 凸包 : ConvexHull -> 凹んだところがない多角形  
(形状は三角形ポリゴンで表される)
  - 三角形ポリゴン :  
BvhTriangleMesh (static object用)  
GImpactTriangleMesh

# 三角形ポリゴン

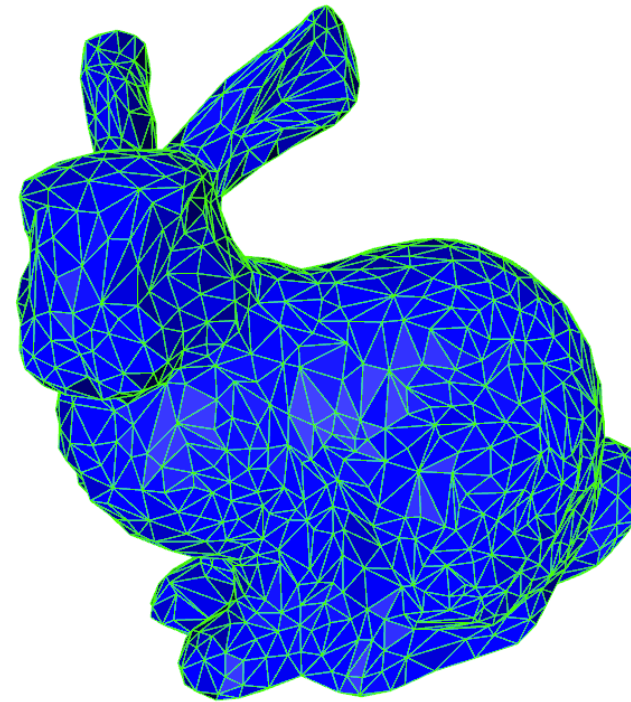
- ポリゴン (polygon)

3次元空間において曲面を多角形の集合により区分線形近似  
3DCGにおいて最も一般的な方式

最もよく使われるのは  
**三角形ポリゴン**



なぜか？

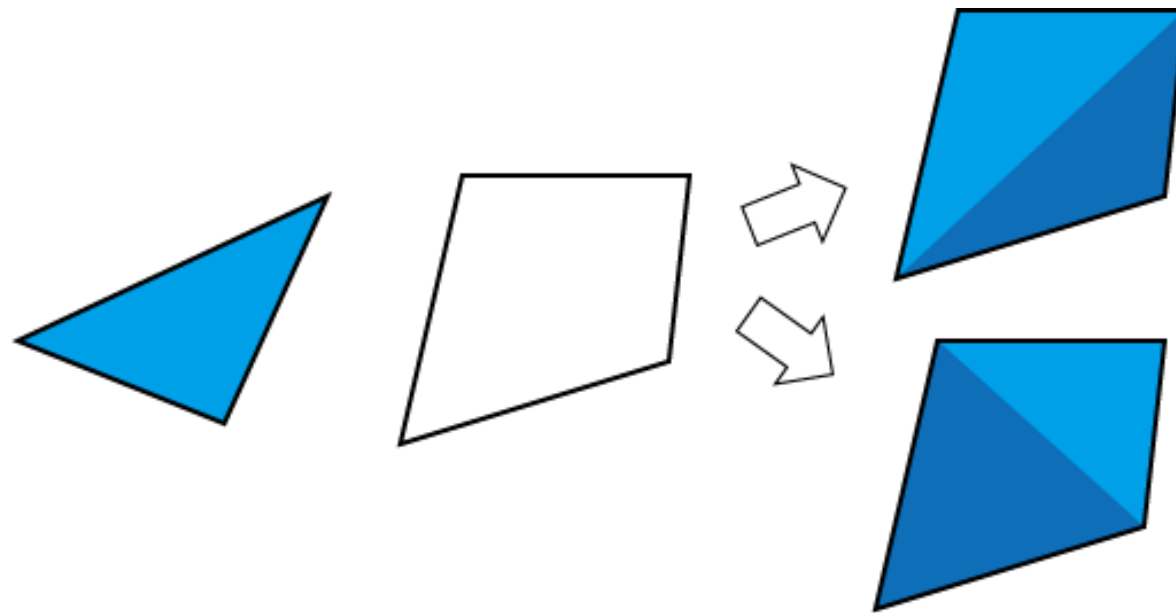


# 三角形ポリゴン

- なぜ三角形ポリゴン？

3点はただ1つの平面を構成する

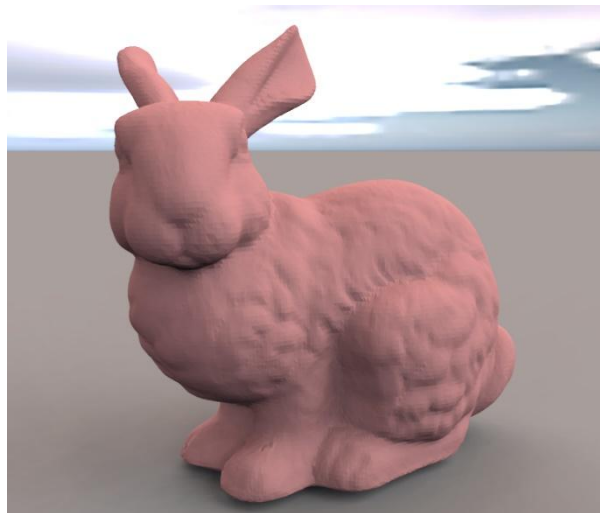
⇒ 四角形以上では平面でない場合もある



四角形や五角形などを三角形に置き換えて描画

# 三角形ポリゴンモデルの例

Stanford Bunny (3MB)  
約3.6万頂点, 7万ポリゴン



Armadillo (7MB)  
約17万頂点, 35万ポリゴン



Asian Dragon (130MB)  
約360万頂点, 720万ポリゴン



The Stanford 3D Scanning Repositoryから取得したデータをレンダリング  
<http://www-graphics.stanford.edu/data/3Dscanrep/>

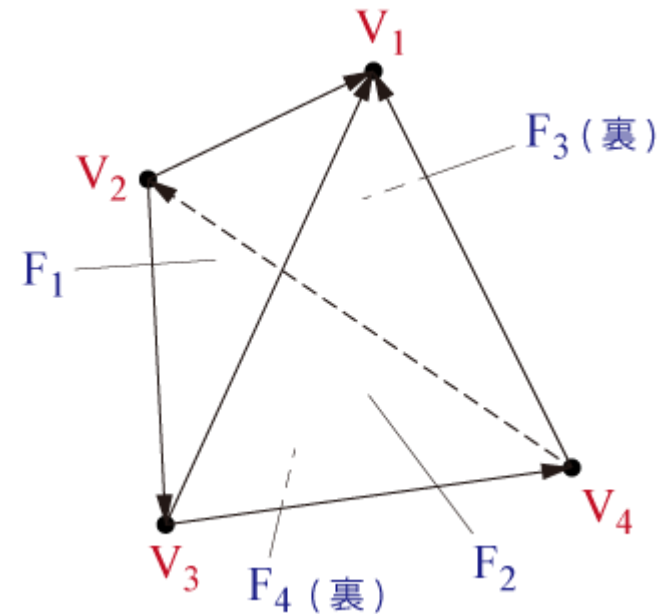
# 三角形ポリゴン

## ■ ポリゴンによる境界表現

ポリゴンデータは一般的に頂点とその接続情報(位相情報)に分けて格納される

頂点データ	面データ(接続情報)
$V_1 : (x_1, y_1, z_1)$	$F_1 : V_1, V_2, V_3$
$V_2 : (x_2, y_2, z_2)$	$F_2 : V_1, V_3, V_4$
$V_3 : (x_3, y_3, z_3)$	$F_3 : V_1, V_4, V_2$
$V_4 : (x_4, y_4, z_4)$	$F_4 : V_2, V_4, V_3$

↑ 表面から見たときに反時計回りになる



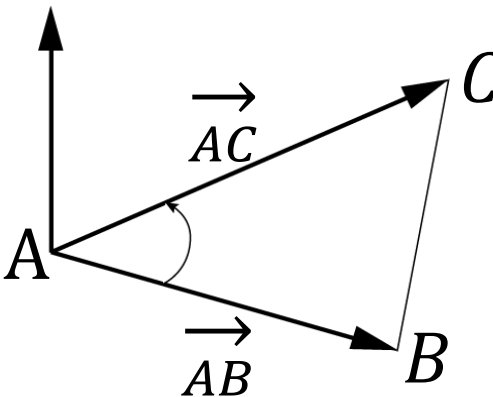
- ・頂点操作による形状変更が容易
- ・周囲の頂点情報から法線などの計算が可能



# 三角形ポリゴン

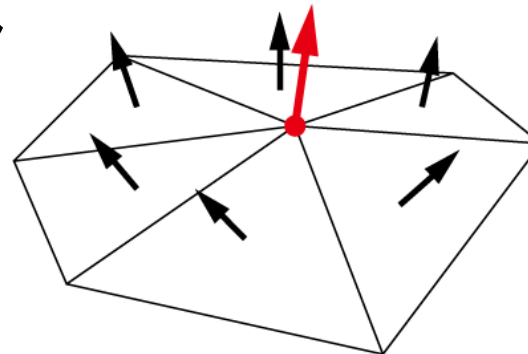
## ■ 法線の計算

法線(その面が向いている方向)はエッジベクトルの外積で計算できる

$$n = \frac{\overrightarrow{AB} \times \overrightarrow{AC}}{|\overrightarrow{AB} \times \overrightarrow{AC}|}$$


反時計回りで表面になる

頂点の法線は周囲のポリゴン  
法線の平均で近似



# 三角形ポリゴン

## ■ ポリゴンによる境界表現

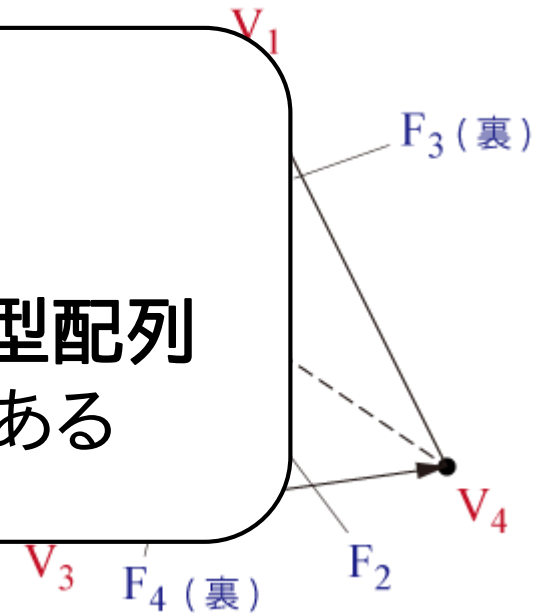
ポリゴンデータは一般的に頂点とその接続情報(位相情報)に分けて格納される

頂点
$V_1 : (x_1, y_1)$
$V_2 : (x_2, y_2)$
$V_3 : (x_3, y_3)$
$V_4 : (x_4, y_4)$

三角形ポリゴンの情報は

- ・頂点座標 : btScalar型配列
  - ・接続情報(インデックス) : int型配列
- に分けてbulletに渡す必要がある

回りになる

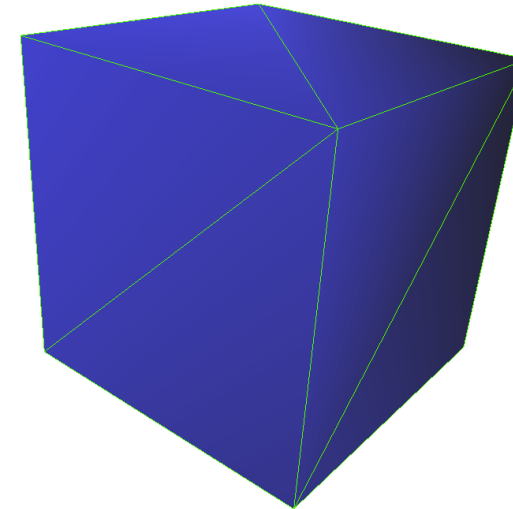
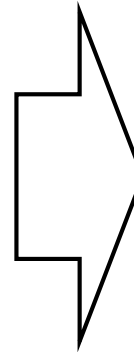


- ・頂点操作による形状変更が容易
- ・周囲の頂点情報から法線などの計算が可能

# 任意形状の読み込み

- 三角形ポリゴンデータ
  - 3Dモデルファイル：三角形ポリゴンの情報をファイルに書き出したもので様々な形式がある(ex. OBJ, 3DS, VRML, STLなど)

頂点情報 {  
g cube  
v -1 1 -1  
v -1 -1 -1  
v 1 -1 -1  
v 1 1 -1  
...  
接続情報 {  
f 1//1 2//2 3//3  
f 1//1 3//3 4//4  
f 5//5 6//6 7//7  
f 5//5 7//7 8//8  
...  
OBJファイルの例



# 任意形状の読み込み

- 三角形ポリゴンデータ

- どうやって読み込むのか？

こちらでOBJファイル読み込むためのコード(ヘッダファイル)を用意してあるのでそれを使ってください。

```
#include "rx_obj.h"

// 3Dファイル読み込み
rxMTL mats;          // 材質
vector<glm::vec3> vrts, nrms; // 頂点座標と法線
vector<rxFace> tris; // 接続情報
rxOBJ obj;
obj.Read(filename, vrts, nrms, tris, mats, true);
```

配列vrtsとtrisに頂点情報と接続情報が入っています

```
// 頂点座標の取り出しの例(btVector3配列への変換)
for(int i = 0; i < (int)vrts.size(); ++i){
    vertices[3*i] = vrts[i][0];
    vertices[3*i+1] = vrts[i][1];
    vertices[3*i+2] = vrts[i][2];
}
```

# 任意形状の読み込み

## ■ Bulletへの3Dモデル登録

これまでの形状のようにいきなりbt\*Shapeを作成するのではなく、先にbullet用のポリゴン配列を作成する必要がある

```
btTriangleIndexVertexArray* tri_array  
    = new btTriangleIndexVertexArray(index_count,  
                                     indices,  
                                     3*sizeof(int),  
                                     vertex_count,  
                                     vertices,  
                                     3*sizeof(btScalar));
```

ポリゴン数  
接続情報配列  
1ポリゴン情報の  
メモリ上のサイズ  
頂点数  
頂点情報配列  
1頂点情報の  
メモリ上のサイズ

indices配列やvertices配列はvrts,trisをそれぞれbtVector3の配列(サイズ=頂点数), int型の配列(三角形数x3の大きさの1次元配列に変換したもの (詳しくは実験Webページ参照))

# 任意形状の読み込み

- Bulletへの3Dモデル登録

bulletのポリゴン情報配列をbtGImpactMeshShapeに渡すことで形状情報を作成

```
#include "BulletCollision/Gimpact/btGImpactCollisionAlgorithm.h"
#include "BulletCollision/Gimpact/btGImpactShape.h"

// ポリゴン形状の登録
btGImpactMeshShape *shape = new btGImpactMeshShape(tri_array);
shape->updateBound(); // 形状境界情報の更新
```

後はこれまでと同じくこの形状情報(shape)でbtRigidBodyを作成して、btDynamicsWorldに登録

# 任意形状の読み込み

- 三角形ポリゴンの描画

専用の描画ルーチンを作る必要がありますが, 配布しているサンプルプログラム2はポリゴン描画も含んでいる(bulletの世界に追加すれば自動で描画される)ので実装の詳細は省略

(詳しく知りたい場合は実験Webページ参照)

# 弾性体

- 弾性体(弾性変形)とは？  
力を加えて変形しても, その力がかからなくなったら完全に元の形状に戻る物質
  - 糸や布, クッションや輪ゴムなど
  - $\Leftrightarrow$  塑性体(塑性変形)
- bulletの弾性体への対応
  - バネ-質点系というモデルで弾性体をシミュレーション
  - ロープ形状, 布形状, 楕円体, 凸包,  
三角形メッシュ, 四面体分割データ(TetGen)



# 弾性体

- 弾性体を使うための準備1

- インクルードファイルの追加

```
#include <BulletSoftBody/btSoftRigidBodyDynamicsWorld.h>
#include <BulletSoftBody/btSoftBodyRigidBodyCollisionConfiguration.h>
#include <BulletSoftBody/btSoftBodyHelpers.h>
#include <BulletSoftBody/btSoftBody.h>
```

サンプルプログラム2だとutils.hで既にインクルードしています

- ライブラリファイルの追加

```
#pragma comment (lib, "BulletSoftBody.lib")
```

Visual Studioならプロジェクトのプロパティ→リンカー→入力→追加の依存ファイルでも追加できます ⇒ サンプルプログラム2では元々設定されています.

macのgmakeの場合は, Makefileに記述しますがこちらもサンプルプログラム2に付属のものは元々記述済み.

# 弾性体

- 弾性体を使うための準備2
  - Bulletワールドの初期化部分の変更

```
// 衝突検出方法の選択(デフォルトを選択)
btDefaultCollisionConfiguration *config
    = new btSoftBodyRigidBodyCollisionConfiguration();
btCollisionDispatcher *dispatcher = new btCollisionDispatcher(config);

// ブロードフェーズ法の設定(Dynamic AABB tree method)
btDbvtBroadphase *broadphase = new btDbvtBroadphase();

// 拘束(剛体間リンク)のソルバ設定
btSequentialImpulseConstraintSolver* solver
    = new btSequentialImpulseConstraintSolver();

// Bulletのワールド作成
g_dynamicsworld = new btSoftRigidDynamicsWorld(dispatcher, broadphase,
                                                solver, config, 0);
```

g\_dynamicsworldの型変更も忘れずに

# 弾性体

- 弾性体を使うための準備3
  - softbodyのための設定

```
btSoftBodyWorldInfo g_softBodyWorldInfo;  
  
...  
  
// btSoftBodyWorldInfoの初期化・設定  
g_softBodyWorldInfo.m_dispatcher = dispatcher;  
g_softBodyWorldInfo.m_broadphase = broadphase;  
g_softBodyWorldInfo.m_sparsesdf.Initialize();  
g_softBodyWorldInfo.m_gravity.setValue(0, -9.8, 0);  
g_softBodyWorldInfo.air_density = 1.2;
```

`g_softBodyWorldInfo`は`btSoftBody`を追加するときに使うので、グローバル変数にしておいてください(Worldには登録しない)

# 弾性体

## ■ 各種弾性体の設定方法

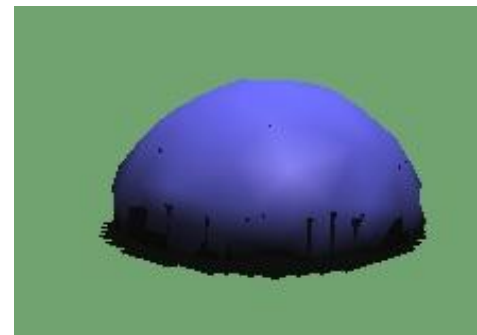
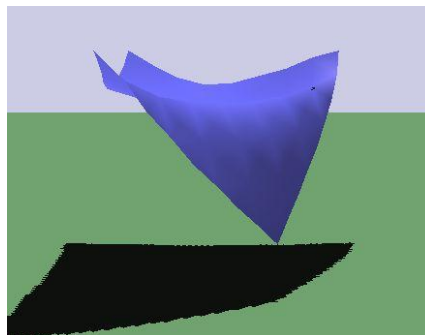
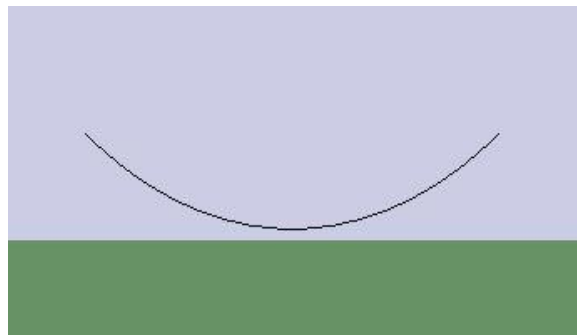
1次元弾性体(ロープ)の作成

```
btSoftBody* rope = btSoftBodyHelpers::CreateRope(g_softBodyWorldInfo,  
                                                    btVector3(-1, 0, 0), // 端点1  
                                                    btVector3( 1, 0, 0), // 端点2  
                                                    16,      // 分割数(ばねの数)  
                                                    1+2); // 端点の固定フラグ  
  
rope->m_cfg.piterations      = 4;      // 最大反復回数  
rope->m_materials[0]->m_kLST = 0.5;    // 剛性 (変形のしやすさ)  
rope->setTotalMass(1.0);          // 全体の質量  
rope->getCollisionShape()->setMargin(0.01);  
g_dynamicsworld->addSoftBody(rope);
```

弾性体の性質設定

弾性体のワールドへの登録

2次元弾性体(布)や3次元弾性体(楕円&三角形メッシュ)の設定については実験Webページを参照



# 任意形状の読み込み

ここからは各自の環境で  
実際に作業

実験ページの「5. 3Dモデル読み込みと弾性体」  
の練習問題を実際にやってみよう  
(余裕のある人はoption課題もやってみよう!)

