情報工学実験３：ゲーム班レポート

平成 18 年 7 月 31 日

ゲーム班メンバー

学籍番号	氏　　名
045727C	杉山千秋
045712E	大城和輝
035736H	仲村章吾
045714B	大城信孝
035713J	小野雅俊

1　目的

PS2Linux を用いてネットワーク型のゲームを作成する。その過程において、限られた資源をいかに活用してゲームを作成するか。限られた機材スペックでいかにして表現を行うか、グループメンバーでいかに協力し合えるかを学ぶ。

2　行った事

- blender でのオブジェクト作成
- blender で作成したオブジェクトを PS2 で表示
- ベクターユニットを用いたオブジェクトの回転やカメラの追跡
- Linda を用いたネットワークの構築
- 先輩のレースゲーム改良

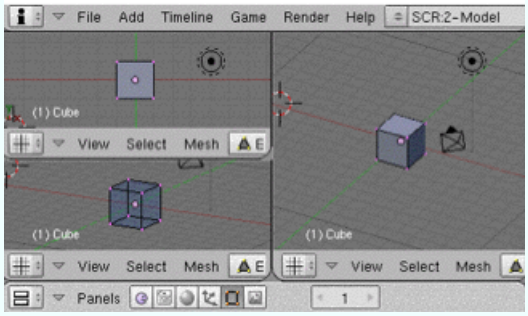
3　作業報告

3.1　Blender

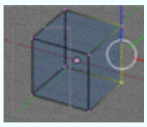
・基本用語

・オブジェクト
Blender上に表示されているもの。
作成したものや、カメラ、ライトがある。
・vertex(頂点)
ポリゴンを構成する点、頂点を動かしてポリゴンを作成する。
・edge(辺)
頂点と頂点をつなぐ辺。
・face(面)
辺で囲まれた平面。面には色をつけたり、画像を貼ったりできる。

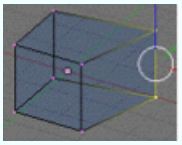
・作成画面



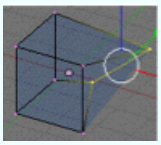
・オブジェクトの編集(基本操作)



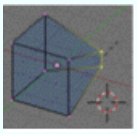
・選択
頂点の選択にはアップルを押しながらクリックする。
複数の頂点を選択する時は、shift+アップルを押しながらクリックする。
一度に複数の頂点を選択する時は、"Bキー"を押すと、選択する範囲を決められる。
"Bキー"を2度押すと、選択する範囲が長方形ではなく円の範囲で選択できる。
円の時は、"ー(マイナス)キー"で円を縮小、"＋キー"で円を拡大できる。
選択された頂点は、ピンクから、黄色に変わり、選択された2頂点を接続する辺は黄色に変わり、選択された辺が囲む面はピンクに変わる。



・移動
頂点を移動するには、あらかじめ頂点を選択し(選択された頂点は黄色で表示される)、
"Gキー"を押すと、頂点移動モードになり、マウスや方向キーで移動することができる。

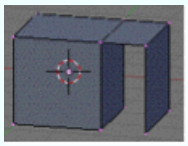


・回転
頂点を回転するには、頂点が選択された状態で、"Rキー"を押す。
回転移動モードになるので、マウスや方向キーを動かすことによって、頂点が回転する。

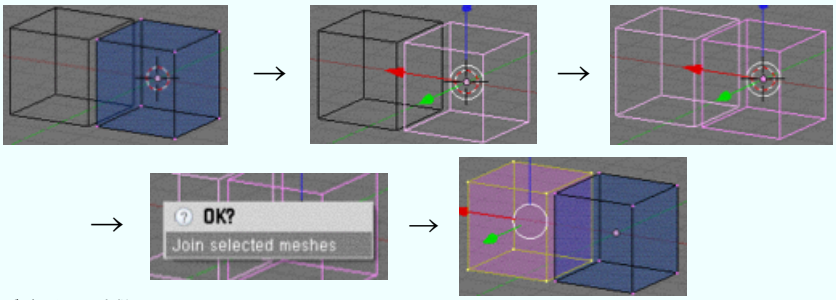


・拡大、縮小
選択した辺や面を拡大縮小するには、"Sキー"を押す。
拡大縮小モードになるので、マウスや方向キーを動かすことによって、
選択した辺、面を拡大、縮小する。

・オブジェクトの編集



・押し出し
選択された頂点や辺、面を押し出すには、"Eキー"を押して、押し出したい範囲
を選択し、マウスや方向キーで押し出します。
Region：選択されたfaceを押し出す。
Only Edges：選択された辺だけを押し出す
Only Vertices：選択された頂点だけを押し出す

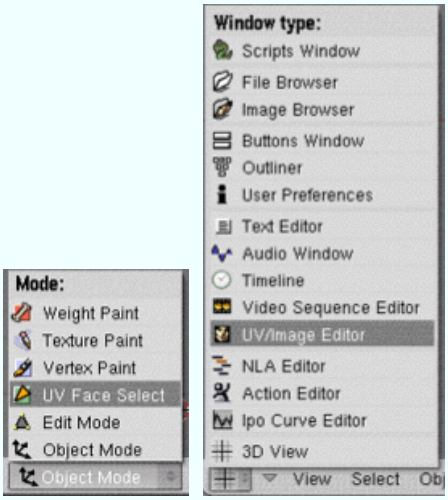


・オブジェクトの合併
複数のオブジェクトをひとつにまとめる時に合併を行う。
まず、オブジェクトモードでひとつのオブジェクトを選択する。その後、shift+アップルを押しながらもうひとつの
オブジェクトをクリックする(もしくは、"Bキー"で選択する)とふたつの選択されたオブジェクトがピンク色に表示される。
合併したいオブジェクトを複数個選択し、ctrl+"Jキー"を押すと、複数のオブジェクトを合併することができる。
Blenderで作成したオブジェクトをPS2で表示するには、この方法でひとつにまとめてからでないとできないため、
この方法をよく使う。

・オブジェクトに色をつける

・オブジェクトに色をつける
オブジェクトに色をつけるには、操作ウインドウでShadingモードにして行う。
Shadingモードを選んだら、Materialの"Add New"を選択すると、RGBバーで色を決めることができる。
上の画像では、R(赤)を1.000、G(緑)を0.291、B(青)を0.279にして、出力している。

・オブジェクトにテクスチャ(画像)を貼る

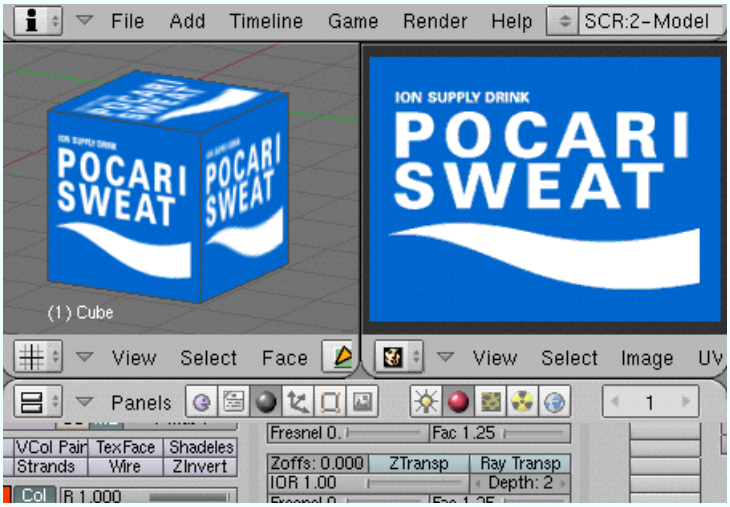


まず、３Dウインドウの画面を2分割以上にしておく。
ひとつの画面を、モード選択で、"UV Face Select"を選択する。

もうひとつの画面のウインドウタイプを
"UV / Image Editor"にしておく。

"UV / Image Editor"のウインドウで、
"Image" → "Open"を選択し、貼りたい画像を選択する。

下の図は、POCARI SWEATのテクスチャを貼った図。



3.2 ベクターユニット

3.2.1 今回頻繁に用いたベクターユニットの関数

- void ps2_vu0_apply_matrix(ps2_vu0_fvector v0, ps2_vu0_fmatrix m0, ps2_vu0_fvector v1)
  マトリックス m0 にベクトル v1 を右から乗算して v0 に与える

$$v0 = m0 * v1$$

- void ps2_vu0_add_vector(ps2_vu0_fvector v0, ps2_vu0_fvector v1, ps2_vu0_fvector v2)
  ベクトル v1 の各要素とベクトル v2 の各要素を各々加算して v0 に与える

$$v0 = v1 + v2$$

- void ps2_vu0_copy_vector(ps2_vu0_fvector v0, ps2_vu0_fvector v1);
  ベクトル v1 をベクトル v2 にコピーする

- void ps2_vu0_unit_matrix(ps2_vu0_fmatrix m0);
  与えられた行列を単位行列に変換する

- void ps2_vu0_rot_matrix_x(ps2_vu0_fmatrix m0, ps2_vu0_fmatrix m1, float rx);
  X 軸を中心とした行列の回転
  回転角 rx より X 軸を中心とした回転マトリックスを求めて、マトリックス m1 に左側から乗算して、その結果をマトリックス m0 に与える。
  X 軸だけでなく Y 軸、Z 軸を中心とした関数もある。また 3 ついっぺんに回転させる関数もある。

3.2.2 ベクターユニットとカメラの勉強として作ったプログラムの説明

この game/ps2/awin は/game/ps2/it_expo_2004 のものを元に作ったプログラムである。

主に加えた改良点は、

- 上下移動の追加
- 弾オブジェクトの発射

- カメラの追跡

である。

```
plane.c:L77~L80
  ps2_vu0_unit_matrix( rot );\\
  ps2_vu0_rot_matrix( rot, rot, obj->angle );
  ps2_vu0_apply_matrix( mov, rot, mov );
  ps2_vu0_add_vector( obj->xyz, obj->xyz, mov );
```

このようにベクターユニットを用いることにより、上下左右の角度変更の後その向きに進行することが可能になっている。
飛行機本体ともう一つ弾用のオブジェクトも同様に設定することで、R 1を押したときに弾オブジェクトだけ進行方向前に飛ばすことができる。これによって弾の発射を再現した。

またカメラの追跡では

```
void
camera_trace_obj( OBJECT *obj )
{
    ps2_vu0_sub_vector( camera.xyz, obj->xyz, camera.zd );
}
```

このような関数を用意した。これは飛行機本体の座標を引数としてもらうことで、飛行機の後ろにカメラが付いていくようにしている。
オブジェクトの xyz 座標から camera.zd を引いた結果を camera.xyz に渡すことで、camera.zd で設定した距離間でカメラはオブジェクトの後ろに付いていく。

3.3 Linda

3.3.1 ネットワーク型のゲームを作成するために使用した Linda についてのまとめ

Linda とは、タプルと呼ばれる ID と DATA がセットになったものを、各クライアントが Linda サーバに対して書き込み、読み込み等を行うことでデータの交換、共有を行い、通信をしているシステムである。以下に Linda の API を示す。

- void start_linda(hostname);
  通信を初期化して、Linda API を hostname で使用可能にする
- int psx_out(int id, char *data, int size);
  data が示すアドレスから、指定した ID のタプルに size byte のデータを書き込むコマンドである。
- int psx_in(int id);
  指定した ID のタプルを読み込み sequence 番号を取得する。指定した ID のタプルはデータを読み込んだ後サーバ上から削除される。
- int psx_rd(int id);
  指定した ID のタプルを読み込み sequence 番号を取得する。指定した ID のタプルはデータを読み込んだ後もサーバ上に残る。
- void psx_sync_n();
  psx_out(),psx_in(),psx_rd() 等のタプル送受信の API を実行した段階ですぐに通信が行われるのではなく、送信タプルは COMMAND キューに、受信タプルは REPLY キューにためられる。この psx_sync_n() はそれをプログラマの好きなタイミングで一気に送受信するための API である。
- unsigned char psx_reply(int seq);
  psx_in() か psx_rd() で取得したシークエンス番号を引数として渡すと、要求したデータが REPLY キューにあればタプルへのポインタを返す。そのポインタには、先頭にタプルのヘッダ情報 12byte が格納されていて、その後にデータが格納されている。
- void psx_free(unsigned char reply) psx_reply() で取得したタプルを消去する。これをしないと REPLY キューにデータが溜まってしまう。

Linda での通信の流れを図１で示す。

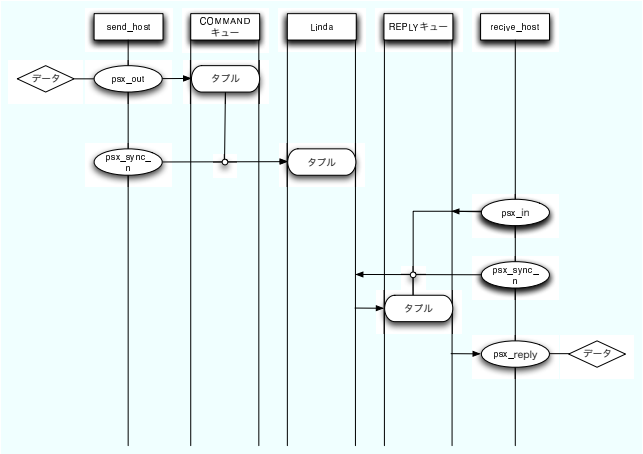


図 1: Linda での通信の流れ

図１からも分かるように実際に Linda にアクセスするのは psx_sync_n() を実行したときだけである。つまり Linda は非同期である。psx_sync_n() が実行されるまで通信は行われないので、相手との同期はとれません。しかしこの方式はプロセッサは待ち状態のときに余分な情報を処理する必要がなくなる。

実際には図１のように送信側と受信側がきっちり分かれるのではなく、全てのクライアントが送受信することになる。

3.3.2 linda 使用時に気をつけること

Linda は非同期なので送信元は受信先が受信したのを確認してから psx_out() を行う必要がある。そうしなければ Linda サーバにタプルが溜まってしまい処理が重くなってしまう。なので受信側も受信したことを示す為のタプルを送り、送信側はそのタプルを psx_in() か psx_rd() で受け取ってから、次の psx_out() をしなければならない。

データが更新されていないのに psx_out() を繰り返してしまうことでも Linda サーバに負荷がかかってしまう。なので以下のようにしてデータが更新されたときだけ psx_out() をする必要がある。

```
  if(key!=old_key){
            psx_out(id,key,sizeof(key));
 }
```

また、psx_reply() が完了したら、psx_free() を実行して REPLY キューにデータが溜まるのを防ぐ必要がある。

3.4 レースゲーム改良点

マゴロクレーシング (五木さんレーシングを改良)

1. コースとマシンの追加
   Blender にて作成したコース及びマシンを追加。コースは、既存のものは小さいのでできるだけコースを大きくし、ビルや建物などクオリティを高くするために力を入れた。また、地面が消えるなどの症状も Blender 上で頂点の数を増やすなどのことにより消えないようにできた。

2. 画像の質向上
   元々の png ファイルの画像は、小さい画像を無理矢理拡大して表示していたため画像がとても荒かった。そのため、png の画像を大きくして拡大しないで表示できるようにした。

3. タイムの導入
   レースゲームには絶対に必要なタイムを導入した。これによりコースを 3 周するのにどれだけ時間がかかったかがわかるようになった。

   時間を取得、時間の形式を変換する関数をまとめて game_time.c に記述した。
   schedule.c からこの関数を呼び出し、3 週するまでの時間を計れるようにしている。

   関数一覧

   `int game_time_get_msec(void)`
   ハードウェアの時間を求めて、ミリ秒形式 (1 / 1000 秒) で値を返す。

   `int game_time_conv_ms2cs (int msec)`
   ミリ秒形式の値から、1/100 秒を求めて返す (2 桁表示)。

   `int game_time_conv_ms2sec (int msec)`
   ミリ秒形式の値から、秒を求めて返す (2 桁表示)。

   `int game_time_conv_ms2minute (int msec)`
   ミリ秒形式の値から、分を求めて返す (2 桁表示)。

```
void game_time_set_raptime (char *s,int msec)
```
引数１に引数２から求めた"XXX:XX:XX"形式の文字列を代入する。

引数２に渡されたミリ秒形式の値を"XXX:XX:XX"形式の文字列に変換。

ポインタを使用して、関数呼び出し時に指定した引数（仮引数）に直接値を代入している。

4. カメラの設定
カメラの位置を見やすい位置に配置し直した。そしてマシンが曲がろうとするときにマシンの横が見えるようにした。これによりリアル感が増した。

```
if(pad.left>0  && game.jiki->speed !=0){
  if(def_camera.angle[1]<=0.2){
  def_camera.angle[1]+=0.02;
  }
}
else if(pad.right>0 && game.jiki->speed !=0){
  if(def_camera.angle[1]>=-0.2){
  def_camera.angle[1]-=0.02;
  }
}
else {
  if(def_camera.angle[1]<-0.01){
  def_camera.angle[1]+=0.02;
  }
  else if(def_camera.angle[1]>0.01){
def_camera.angle[1]-=0.02;
  }
```

pad.left が 0 より大きくて (左キーが入力されている場合) スピードが 0 でないときカメラのアングルが y 軸に対して 0.2 以上にならないように動く。左キーが離されると徐々にカメラのアングルが戻る。右キーの場合も同様である。

5. 光源の設定
光源の位置を配置し直したのでコースが全体的に明るく見えるようになった。

4　今後の課題

- コースの拡大
- Linda での通信対戦時の処理を軽くする
- 通信対戦時でのライバルとの当たり判定
- 通信大戦時でのスタートの同期
- 車とコースの選択画面の改良
- 坂道でもスムーズに走れるようにする
- タイトル画面に戻ろうとするときに起きる Segmentation fault をなくす