

プログラミング演習

—ミニゲーム—

学籍番号：16426
4年 電子情報工学科 23番
福澤 大地

提出日：2020年2月3日

1 目的

OpenGL (Open Graphics Library) に準拠した C 言語のライブラリ “GLUT (OpenGL Utility Toolkit)” を用いてミニゲームのプログラムを作成することで、グラフィカルなウィンドウアプリケーションを作成できるようになる。また、コールバック関数を用いたイベント駆動型プログラミングを行うことで、インターフェイスを実現する。

2 作成したミニゲーム

ビリヤードのナインボールを行うゲームを作成した。OpenGL の 3D 描画機能を利用して、ボールを立体的に表示するなど、リアリティを追求した。

また、簡易的な AI を実装することで、人 vs 人か、人 vs コンピュータを選択できるようにした。

3 開発環境

プログラムの開発、実行を行った環境を表 1 に示す。

表 1 開発環境

CPU	Intel Core i5-7400 @ 3.0GHz
メモリ	8GB
OS	Microsoft Windows 10 Home
システム	64bit
実行環境	Cygwin 3.0.7
コンパイラ	GCC 7.4.0
OpenGL ライブラリ	GLUT 3.7

4 OpenGL と GLUT

OpenGL とは、2 次元/3 次元のコンピュータグラフィックライブラリである。幅広い処理系に対応しており、汎用性が高いため、広く普及している。

一方で、簡単な機能しか用意されていない複雑な処理は向いていないため、それを補うために登場したのが GLU (OpenGL Utility Library) である。GLU は OpenGL の補助的なライブラリであり、円柱などの複雑な図形や、テクスチャの処理といった機能を提供する。GLU の機能に加え、ウィンドウ作成やイベント処理などの機能を提供し、より高度なグラフィックを作成することを可能にしたものが今回使用する GLUT である [1]。

5 プログラムリスト

プログラムのソースコードを、リスト 1-10 に、make 時のルールを記述した Makefile をリスト 11 に示す。

リスト 1 billiard.h

```
1 #ifndef BILLIARD_H
2 #define BILLIARD_H
```

```

3
4 #include <GL/glut.h>
5 #include <GL/glpng.h>
6
7 #include "ball.h"
8 #include "vector.h"
9
10 #define BALL_NUM 10 // 球の数
11 #define TURN_TIME 120 // ターン表示の時間
12 #define RESULT_TIME 240 // リザルト表示の時間
13
14 // キュー
15 struct cue {
16     struct vector p; // 位置
17     double angle; // 角度
18     double power; // 力
19     GLuint image; // 画像
20 };
21
22 // シーン
23 enum scene {
24     Title, // タイトル
25     Game, // ゲーム画面
26 };
27
28 // 状態
29 enum status {
30     Stop, // 静止中
31     Pull, // キューを引いてる
32     Put, // 手球を配置中
33     Move, // 移動中
34     Result // 勝者表示
35 };
36
37 // ターン
38 enum turn {
39     Player1,
40     Player2,
41     Player,
42     CPU
43 };
44
45 void init(void);
46 void update(void);
47
48 int ballMoving(void);
49 int canPut(void);
50 struct vector convertPoint(int, int);
51
52 void Display(void);
53 void Reshape(int, int);
54 void Timer(int);
55 void Mouse(int, int, int, int);
56 void PassiveMotion(int, int);
57
58 extern struct ball prev_balls[BALL_NUM];

```

```

59 extern struct ball balls[BALL_NUM];
60 extern struct table table;
61 extern struct cue cue;
62
63 extern GLuint title_image;
64 extern GLuint vshuman_images[2];
65 extern GLuint hscpu_images[2];
66 extern GLuint invalid_image;
67 extern GLuint highlight_image;
68 extern GLuint turn_images[4];
69 extern GLuint win_images[4];
70
71 extern struct vector mouse;
72 extern enum scene scene;
73 extern enum status status;
74 extern enum turn turn;
75 extern int turn_left;
76 extern int win_left;
77 extern int break_shot;
78 extern int next;
79 extern int first_touch;
80
81 #endif

```

リスト 2 billiard.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <time.h>
5
6 #include <GL/glut.h>
7 #include <GL/glpng.h>
8
9 #include "billiard.h"
10 #include "table.h"
11 #include "ball.h"
12 #include "vector.h"
13 #include "shape.h"
14
15 #define FPS 60 // フレームレート
16 #define ASPECT 2.0 // アスペクト比 (幅/高さ)
17 #define BALL_R 0.04 // 球の半径
18 #define CUE_W 1.536 // キューの幅
19 #define CUE_H 0.048 // キューの高さ
20
21 // 角度を変換
22 #define radian(deg) (rad * M_PI / 180.0)
23 #define degree(rad) (rad * 180.0 / M_PI)
24
25 // オブジェクト
26 struct ball prev_balls[BALL_NUM]; // 前回の球
27 struct ball balls[BALL_NUM]; // 球
28 struct table table = {{1.75, 0.875}, 0.0896, 0.8}; // 台
29 struct cue cue; // キュー
30

```

```

31 // 画像
32 GLuint title_image;           // タイトル
33 GLuint vshuman_images[2];    // 人vs人のボタン
34 GLuint vscpu_images[2];     // 人vsCPUのボタン
35 GLuint invalid_image;       // 無効マーク
36 GLuint highlight_image;     // ハイライト
37 GLuint turn_images[4];      // ターン表示
38 GLuint win_images[4];       // 勝者表示
39
40 struct vector mouse = {ASPECT, 1}; // マウス座標
41 enum scene scene = Title;        // シーン
42 enum status status;            // 状態
43 enum turn turn;               // ターン
44 int turn_left;                // ターン表示の残り時間
45 int win_left;                 // 勝者表示の残り時間
46 int break_shot;               // ブレイクショットか
47 int next;                     // 次に狙う球
48 int first_touch;              // 最初に当たった球
49
50 int main(int argc, char *argv[]) {
51     int i;
52     char fileName[FILENAME_MAX];
53
54     srand(time(NULL));
55
56     // 初期化
57     glutInit(&argc, argv);
58     glutInitWindowSize(960, 480);
59     glutCreateWindow("REAL BILLIARD");
60     glutInitDisplayMode(GLUT_RGBA);
61     glClearColor(1.0, 1.0, 1.0, 1.0);
62
63     // 混合処理を有効化
64     glEnable(GL_BLEND);
65     glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
66
67     // 線のアンチエイリアスを有効化
68     glEnable(GL_LINE_SMOOTH);
69     glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);
70
71     // 陰影を有効化
72     glEnable(GL_LIGHTING);
73     glEnable(GL_LIGHT0);
74     glEnable(GL_LIGHT1);
75     glEnable(GL_CULL_FACE);
76
77     // コールバック関数登録
78     glutDisplayFunc(Display);
79     glutReshapeFunc(Reshape);
80     glutTimerFunc(1000.0 / FPS + 0.5, Timer, 0);
81     glutMouseFunc(Mouse);
82     glutPassiveMotionFunc(PassiveMotion);
83
84     // 画像読み込み
85     title_image = pngBind("images/title.png", PNG_NOMIPMAP, PNG_ALPHA, NULL, GL_CLAMP,
GL_NEAREST, GL_NEAREST);

```

```

86
87     for (i = 0; i < 2; i++) {
88         sprintf(fileName, "images/vshuman_%d.png", i);
89         vshuman_images[i] = pngBind(fileName, PNG_NOMIPMAP, PNG_ALPHA, NULL, GL_CLAMP,
90                                     GL_NEAREST, GL_NEAREST);
91
92         sprintf(fileName, "images/vscpu_%d.png", i);
93         vscpu_images[i] = pngBind(fileName, PNG_NOMIPMAP, PNG_ALPHA, NULL, GL_CLAMP,
94                                     GL_NEAREST, GL_NEAREST);
95     }
96
97     for (i = 0; i < BALL_NUM; i++) {
98         sprintf(fileName, "images/ball_%d.png", i);
99         balls[i].image = pngBind(fileName, PNG_NOMIPMAP, PNG_ALPHA, NULL, GL_CLAMP,
100                                GL_NEAREST, GL_NEAREST);
101
102     }
103
104     table.image = pngBind("images/table.png", PNG_NOMIPMAP, PNG_ALPHA, NULL, GL_CLAMP,
105                            GL_NEAREST, GL_NEAREST);
106     cue.image = pngBind("images/cue.png", PNG_NOMIPMAP, PNG_ALPHA, NULL, GL_CLAMP,
107                          GL_NEAREST, GL_NEAREST);
108     invalid_image = pngBind("images/invalid.png", PNG_NOMIPMAP, PNG_ALPHA, NULL, GL_CLAMP,
109                             GL_NEAREST, GL_NEAREST);
110     highlight_image = pngBind("images/highlight.png", PNG_NOMIPMAP, PNG_ALPHA, NULL,
111                               GL_CLAMP, GL_NEAREST, GL_NEAREST);
112
113     // メインループ開始
114     glutMainLoop();
115
116     return 0;
117 }
118
119 // 初期化
120 void init(void) {
121     struct vector p1 = {table.size.x / 2, 0}; // 1番玉の位置
122     double r = BALL_R + 0.0001;           // 球の間隔
123
124     // 球を配置
125     initBall(&balls[0], 0, vector(-table.size.x / 2, 0), BALL_R);
126     initBall(&balls[1], 1, vector(p1.x, p1.y), BALL_R);
127     initBall(&balls[2], 2, vector(p1.x + r * 2 * sqrt(3), p1.y - r * 2), BALL_R);
128     initBall(&balls[3], 3, vector(p1.x + r * 4 * sqrt(3), p1.y), BALL_R);
129     initBall(&balls[4], 4, vector(p1.x + r * 2 * sqrt(3), p1.y + r * 2), BALL_R);
130     initBall(&balls[5], 5, vector(p1.x + r * sqrt(3), p1.y - r), BALL_R);
131     initBall(&balls[6], 6, vector(p1.x + r * sqrt(3), p1.y + r), BALL_R);
132     initBall(&balls[7], 7, vector(p1.x + r * 3 * sqrt(3), p1.y - r), BALL_R);

```

```

133     initBall(&balls[8], 8, vector(p1.x + r * 3 * sqrt(3), p1.y + r), BALL_R);
134     initBall(&balls[9], 9, vector(p1.x + r * 2 * sqrt(3), p1.y), BALL_R);
135
136     // キューを配置
137     cue.p = balls[0].p;
138     cue.angle = angle(sub(mouse, balls[0].p));
139     cue.power = 0;
140 }
141
142 // 更新
143 void update(void) {
144     int i, j;
145
146     // 表示タイマーを更新
147     if (status == Result) {
148         if (win_left > 0)
149             win_left--;
150         else
151             scene = Title;
152     } else {
153         if (turn_left > 0)
154             turn_left--;
155     }
156
157     switch (status) {
158         // 静止中
159         case Stop: {
160             if (turn == CPU) {
161                 int target = 0;
162                 double angle_min = 0;
163
164                 // ポケットの座標
165                 struct vector pockets[6] = {
166                     {-table.size.x, table.size.y},
167                     {0, table.size.y},
168                     {table.size.x, table.size.y},
169                     {-table.size.x, -table.size.y},
170                     {0, -table.size.y},
171                     {table.size.x, -table.size.y}
172                 };
173
174                 // 一番入れやすいポケットを狙う
175                 for (i = 0; i < 6; i++) {
176                     double angle = cos(angle2(sub(pockets[i], balls[next].p), sub(balls[0].
177                         p, balls[next].p)));
178
179                     if (angle < angle_min) {
180                         angle_min = angle;
181                         target = i;
182                     }
183                 }
184
185                 cue.angle = angle(sub(add(balls[next].p, mult(normal(sub(balls[next].p,
pockets[target])), balls[0].r + balls[next].r)), balls[0].p)) + ((double)rand() /
RAND_MAX - 0.5) * 0.01;
status = Pull;

```

```

186         } else {
187             // キューをマウスの方向に向ける
188             cue.angle = angle(sub(mouse, balls[0].p));
189         }
190
191         break;
192     }
193
194     // 手球を配置中
195     case Put: {
196         if (turn == CPU) {
197             // 置ける場所にランダムに配置
198             do {
199                 balls[0].p = vector((2.0 * rand() / RAND_MAX - 1) * table.size.x, (2.0
200 * rand() / RAND_MAX - 1) * table.size.y);
201             } while (!canPut());
202
203             cue.p = balls[0].p;
204             status = Stop;
205         } else {
206             balls[0].p = mouse;
207         }
208
209         break;
210     }
211
212     // キューを引いている
213     case Pull: {
214         // 力を溜める
215         cue.power += 0.001;
216         if (cue.power > 0.12)
217             cue.power = 0.12;
218
219         if (turn == CPU && cue.power >= 0.08) {
220             // 状態を保持
221             for (i = 0; i < BALL_NUM; i++)
222                 prev_balls[i] = balls[i];
223
224             // 手球に初速度を与える
225             balls[0].v = mult(vector(cos(cue.angle), sin(cue.angle)), cue.power);
226             cue.power = 0;
227             first_touch = 0;
228             status = Move;
229         }
230
231         break;
232     }
233
234     case Move: {
235         // 移動
236         for (i = 0; i < BALL_NUM; i++) {
237             if (balls[i].exist)
238                 moveBall(&balls[i]);
239         }
240
241         // 最初に触れた球

```

```

241         if (!first_touch) {
242             for (i = 1; i < BALL_NUM; i++) {
243                 if (balls[i].exist && ballColliding(balls[0], balls[i])) {
244                     first_touch = balls[i].num;
245                     break;
246                 }
247             }
248         }
249
250         // 球の反射
251         for (i = 0; i < BALL_NUM; i++) {
252             if (!balls[i].exist) continue;
253
254             // 球同士の反射
255             for (j = i + 1; j < BALL_NUM; j++) {
256                 if (balls[j].exist)
257                     reflectBall(&balls[i], &balls[j], break_shot);
258             }
259
260             // 台との反射
261             reflectTable(table, &balls[i]);
262         }
263
264         // ポケットイン
265         for (i = 0; i < BALL_NUM; i++) {
266             if (balls[i].exist)
267                 pocketIn(table, &balls[i]);
268         }
269
270         // 球が止まったら
271         if (!ballMoving()) {
272             // ファウルしていなかったら
273             if (balls[0].exist && first_touch == next) {
274                 // 9番球が入っていなかったら
275                 if (balls[9].exist) {
276                     int pocket = 0;
277                     status = Stop;
278
279                     // いずれかの球が落ちていたら続行
280                     for (i = 0; i < BALL_NUM; i++) {
281                         if (prev_balls[i].exist && !balls[i].exist) {
282                             pocket = 1;
283                             break;
284                         }
285                     }
286
287                     // 球が落ちていなかったらターンチェンジ
288                     if (!pocket) {
289                         turn ^= 1;
290                         turn_left = TURN_TIME;
291                     }
292
293                     break_shot = 0;
294                     cue.p = balls[0].p;
295                     cue.angle = angle(sub(mouse, balls[0].p));
296                 } else {

```

```

297 // 9番球が入ったら勝者表示
298     status = Result;
299     win_left = RESULT_TIME;
300 }
301 } else {
302     // ファウルだったらターンチェンジして手球の自由配置
303     status = Put;
304     turn ^= 1;
305     turn_left = TURN_TIME;
306
307     // 球の状態を復元
308     for (i = 0; i < BALL_NUM; i++)
309         balls[i] = prev_balls[i];
310
311     balls[0].p = mouse;
312     balls[0].angle = 0;
313 }
314
315 // 次に落とすべき球
316 next = 0;
317 for (i = 1; i < BALL_NUM; i++) {
318     if (balls[i].exist && (balls[i].num < next || next == 0))
319         next = balls[i].num;
320 }
321
322     break;
323 }
324
325 default:
326     break;
327 }
328 }
329 }
330
331 // 球が動いているか
332 int ballMoving(void) {
333     int i;
334
335     for (i = 0; i < BALL_NUM; i++) {
336         if (!isZero(balls[i].v))
337             return 1;
338     }
339
340     return 0;
341 }
342
343 // 球を置けるか
344 int canPut(void) {
345     int i;
346
347     if (fabs(balls[0].p.x) > table.size.x - balls[0].r || fabs(balls[0].p.y) > table.size.y
348         - balls[0].r)
349         return 0;
350
351     if (pocketTouching(table, balls[0]))
352         return 0;

```

```

352
353     for (i = 1; i < BALL_NUM; i++) {
354         if (balls[i].exist && ballColliding(balls[0], balls[i]))
355             return 0;
356     }
357
358     return 1;
359 }
360
361 // ピクセル座標を描画時の座標に変換
362 struct vector convertPoint(int x, int y) {
363     int w = glutGet(GLUT_WINDOW_WIDTH);
364     int h = glutGet(GLUT_WINDOW_HEIGHT);
365
366     // ウィンドウの縦横比
367     double ratio = (double)w / h;
368
369     // 座標を変換
370     if (ratio > ASPECT)
371         return vector(((double)x / w - 0.5) * 2 * ratio, ((double)y / h - 0.5) * -2);
372     else
373         return vector(((double)x / w - 0.5) * 2 * ASPECT, ((double)y / h - 0.5) * 2 *
374             ASPECT / -ratio);
374 }
375
376 // 画面描画
377 void Display(void) {
378     int i;
379     int target = 0;
380     double predict_min;
381     struct vector predict_pos;
382
383     // 光源の座標
384     GLfloat lightPos[2][4] = {
385         {-1.0, 0.0, 10.0, 1.0},
386         {1.0, 0.0, 10.0, 1.0}
387     };
388
389     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
390
391     if (scene == Title) {
392         // タイトル画面
393         putSprite(title_image, 0, 0, ASPECT * 2, 2, 1);
394
395         if (fabs(mouse.x) <= ASPECT / 2 && -0.25 <= mouse.y && mouse.y <= 0.25)
396             putSprite(vshuman_images[1], 0, 0, ASPECT, 0.5, 1);
397         else
398             putSprite(vshuman_images[0], 0, 0, ASPECT, 0.5, 1);
399
400         if (fabs(mouse.x) <= ASPECT / 2 && -0.875 <= mouse.y && mouse.y <= -0.375)
401             putSprite(vscpu_images[1], 0, -0.625, ASPECT, 0.5, 1);
402         else
403             putSprite(vscpu_images[0], 0, -0.625, ASPECT, 0.5, 1);
404     } else {
405         // 光源を設定
406         glLightfv(GL_LIGHT0, GL_POSITION, lightPos[0]);

```

```

407     glLightfv(GL_LIGHT1, GL_POSITION, lightPos[1]);
408
409     // 台
410     putSprite(table.image, 0, 0, ASPECT * 2, 2, 1);
411
412     // ハイライト
413     if (status == Stop || status == Pull || status == Put) {
414         for (i = 1; i < BALL_NUM; i++) {
415             if (balls[i].exist && next == balls[i].num) {
416                 putSprite(highlight_image, balls[i].p.x, balls[i].p.y, balls[i].r * 3,
417 balls[i].r * 3, 1);
418                 break;
419             }
420         }
421
422     // 予測線
423     if (status == Stop || status == Pull) {
424         // 台との接触
425         if (sin(cue.angle) > 0)
426             predict_pos = vector(balls[0].p.x - (balls[0].p.y - table.size.y + balls
427 [0].r) / tan(cue.angle), table.size.y - balls[0].r);
428         else
429             predict_pos = vector(balls[0].p.x - (balls[0].p.y + table.size.y - balls
430 [0].r) / tan(cue.angle), balls[0].r - table.size.y);
431
432         if (fabs(predict_pos.x) < table.size.x - balls[0].r)
433             predict_min = dist(balls[0].p, predict_pos);
434
435         if (cos(cue.angle) > 0)
436             predict_pos = vector(table.size.x - balls[0].r, balls[0].p.y - tan(cue.
437 angle) * (balls[0].p.x - table.size.x + balls[0].r));
438         else
439             predict_pos = vector(balls[0].r - table.size.x, balls[0].p.y - tan(cue.
440 angle) * (balls[0].p.x + table.size.x - balls[0].r));
441
442         if (fabs(predict_pos.y) < table.size.y - balls[0].r)
443             predict_min = dist(balls[0].p, predict_pos);
444
445     // 球との接触
446     for (i = 1; i < BALL_NUM; i++) {
447         double touch;
448         double predict_dist;
449
450         if (!balls[i].exist || cos(angle(sub(balls[i].p, balls[0].p)) - cue.angle)
451 < 0) continue;
452
453         // 手球の軌道上との距離を算出
454         touch = fabs(tan(cue.angle) * (balls[i].p.x - balls[0].p.x) - balls[i].p.y
455 + balls[0].p.y) / mag(vector(tan(cue.angle), 1));
456
457         if (touch < balls[0].r + balls[i].r) {
458             // 衝突予想地点を算出
459             predict_dist = dist(balls[0].p, balls[i].p) * cos(angle(sub(balls[i].p,
460 balls[0].p)) - cue.angle) - sqrt(pow(balls[0].r + balls[i].r, 2) - pow(touch, 2));
461
462         }
463     }
464
465     if (predict_min < touch)
466         predict_pos = vector(balls[0].p.x - touch * cos(cue.angle),
467 balls[0].p.y - touch * sin(cue.angle));
468
469     if (predict_min < predict_dist)
470         predict_pos = vector(balls[0].p.x - predict_dist * cos(cue.angle),
471 balls[0].p.y - predict_dist * sin(cue.angle));
472
473     if (predict_min < predict_min)
474         predict_pos = vector(balls[0].p.x - predict_min * cos(cue.angle),
475 balls[0].p.y - predict_min * sin(cue.angle));
476
477     if (predict_pos.x < 0)
478         predict_pos.x = 0;
479
480     if (predict_pos.y < 0)
481         predict_pos.y = 0;
482
483     if (predict_pos.x > table.size.x)
484         predict_pos.x = table.size.x;
485
486     if (predict_pos.y > table.size.y)
487         predict_pos.y = table.size.y;
488
489     if (predict_pos.x < balls[0].p.x)
490         predict_pos.x = balls[0].p.x;
491
492     if (predict_pos.y < balls[0].p.y)
493         predict_pos.y = balls[0].p.y;
494
495     if (predict_pos.x > balls[0].p.x)
496         predict_pos.x = balls[0].p.x;
497
498     if (predict_pos.y > balls[0].p.y)
499         predict_pos.y = balls[0].p.y;
500
501     if (predict_pos.x < 0)
502         predict_pos.x = 0;
503
504     if (predict_pos.y < 0)
505         predict_pos.y = 0;
506
507     if (predict_pos.x > table.size.x)
508         predict_pos.x = table.size.x;
509
510     if (predict_pos.y > table.size.y)
511         predict_pos.y = table.size.y;
512
513     if (predict_pos.x < balls[0].p.x)
514         predict_pos.x = balls[0].p.x;
515
516     if (predict_pos.y < balls[0].p.y)
517         predict_pos.y = balls[0].p.y;
518
519     if (predict_pos.x > balls[0].p.x)
520         predict_pos.x = balls[0].p.x;
521
522     if (predict_pos.y > balls[0].p.y)
523         predict_pos.y = balls[0].p.y;
524
525     if (predict_pos.x < 0)
526         predict_pos.x = 0;
527
528     if (predict_pos.y < 0)
529         predict_pos.y = 0;
530
531     if (predict_pos.x > table.size.x)
532         predict_pos.x = table.size.x;
533
534     if (predict_pos.y > table.size.y)
535         predict_pos.y = table.size.y;
536
537     if (predict_pos.x < balls[0].p.x)
538         predict_pos.x = balls[0].p.x;
539
540     if (predict_pos.y < balls[0].p.y)
541         predict_pos.y = balls[0].p.y;
542
543     if (predict_pos.x > balls[0].p.x)
544         predict_pos.x = balls[0].p.x;
545
546     if (predict_pos.y > balls[0].p.y)
547         predict_pos.y = balls[0].p.y;
548
549     if (predict_pos.x < 0)
550         predict_pos.x = 0;
551
552     if (predict_pos.y < 0)
553         predict_pos.y = 0;
554
555     if (predict_pos.x > table.size.x)
556         predict_pos.x = table.size.x;
557
558     if (predict_pos.y > table.size.y)
559         predict_pos.y = table.size.y;
560
561     if (predict_pos.x < balls[0].p.x)
562         predict_pos.x = balls[0].p.x;
563
564     if (predict_pos.y < balls[0].p.y)
565         predict_pos.y = balls[0].p.y;
566
567     if (predict_pos.x > balls[0].p.x)
568         predict_pos.x = balls[0].p.x;
569
570     if (predict_pos.y > balls[0].p.y)
571         predict_pos.y = balls[0].p.y;
572
573     if (predict_pos.x < 0)
574         predict_pos.x = 0;
575
576     if (predict_pos.y < 0)
577         predict_pos.y = 0;
578
579     if (predict_pos.x > table.size.x)
580         predict_pos.x = table.size.x;
581
582     if (predict_pos.y > table.size.y)
583         predict_pos.y = table.size.y;
584
585     if (predict_pos.x < balls[0].p.x)
586         predict_pos.x = balls[0].p.x;
587
588     if (predict_pos.y < balls[0].p.y)
589         predict_pos.y = balls[0].p.y;
590
591     if (predict_pos.x > balls[0].p.x)
592         predict_pos.x = balls[0].p.x;
593
594     if (predict_pos.y > balls[0].p.y)
595         predict_pos.y = balls[0].p.y;
596
597     if (predict_pos.x < 0)
598         predict_pos.x = 0;
599
600     if (predict_pos.y < 0)
601         predict_pos.y = 0;
602
603     if (predict_pos.x > table.size.x)
604         predict_pos.x = table.size.x;
605
606     if (predict_pos.y > table.size.y)
607         predict_pos.y = table.size.y;
608
609     if (predict_pos.x < balls[0].p.x)
610         predict_pos.x = balls[0].p.x;
611
612     if (predict_pos.y < balls[0].p.y)
613         predict_pos.y = balls[0].p.y;
614
615     if (predict_pos.x > balls[0].p.x)
616         predict_pos.x = balls[0].p.x;
617
618     if (predict_pos.y > balls[0].p.y)
619         predict_pos.y = balls[0].p.y;
620
621     if (predict_pos.x < 0)
622         predict_pos.x = 0;
623
624     if (predict_pos.y < 0)
625         predict_pos.y = 0;
626
627     if (predict_pos.x > table.size.x)
628         predict_pos.x = table.size.x;
629
630     if (predict_pos.y > table.size.y)
631         predict_pos.y = table.size.y;
632
633     if (predict_pos.x < balls[0].p.x)
634         predict_pos.x = balls[0].p.x;
635
636     if (predict_pos.y < balls[0].p.y)
637         predict_pos.y = balls[0].p.y;
638
639     if (predict_pos.x > balls[0].p.x)
640         predict_pos.x = balls[0].p.x;
641
642     if (predict_pos.y > balls[0].p.y)
643         predict_pos.y = balls[0].p.y;
644
645     if (predict_pos.x < 0)
646         predict_pos.x = 0;
647
648     if (predict_pos.y < 0)
649         predict_pos.y = 0;
650
651     if (predict_pos.x > table.size.x)
652         predict_pos.x = table.size.x;
653
654     if (predict_pos.y > table.size.y)
655         predict_pos.y = table.size.y;
656
657     if (predict_pos.x < balls[0].p.x)
658         predict_pos.x = balls[0].p.x;
659
660     if (predict_pos.y < balls[0].p.y)
661         predict_pos.y = balls[0].p.y;
662
663     if (predict_pos.x > balls[0].p.x)
664         predict_pos.x = balls[0].p.x;
665
666     if (predict_pos.y > balls[0].p.y)
667         predict_pos.y = balls[0].p.y;
668
669     if (predict_pos.x < 0)
670         predict_pos.x = 0;
671
672     if (predict_pos.y < 0)
673         predict_pos.y = 0;
674
675     if (predict_pos.x > table.size.x)
676         predict_pos.x = table.size.x;
677
678     if (predict_pos.y > table.size.y)
679         predict_pos.y = table.size.y;
680
681     if (predict_pos.x < balls[0].p.x)
682         predict_pos.x = balls[0].p.x;
683
684     if (predict_pos.y < balls[0].p.y)
685         predict_pos.y = balls[0].p.y;
686
687     if (predict_pos.x > balls[0].p.x)
688         predict_pos.x = balls[0].p.x;
689
690     if (predict_pos.y > balls[0].p.y)
691         predict_pos.y = balls[0].p.y;
692
693     if (predict_pos.x < 0)
694         predict_pos.x = 0;
695
696     if (predict_pos.y < 0)
697         predict_pos.y = 0;
698
699     if (predict_pos.x > table.size.x)
700         predict_pos.x = table.size.x;
701
702     if (predict_pos.y > table.size.y)
703         predict_pos.y = table.size.y;
704
705     if (predict_pos.x < balls[0].p.x)
706         predict_pos.x = balls[0].p.x;
707
708     if (predict_pos.y < balls[0].p.y)
709         predict_pos.y = balls[0].p.y;
710
711     if (predict_pos.x > balls[0].p.x)
712         predict_pos.x = balls[0].p.x;
713
714     if (predict_pos.y > balls[0].p.y)
715         predict_pos.y = balls[0].p.y;
716
717     if (predict_pos.x < 0)
718         predict_pos.x = 0;
719
720     if (predict_pos.y < 0)
721         predict_pos.y = 0;
722
723     if (predict_pos.x > table.size.x)
724         predict_pos.x = table.size.x;
725
726     if (predict_pos.y > table.size.y)
727         predict_pos.y = table.size.y;
728
729     if (predict_pos.x < balls[0].p.x)
730         predict_pos.x = balls[0].p.x;
731
732     if (predict_pos.y < balls[0].p.y)
733         predict_pos.y = balls[0].p.y;
734
735     if (predict_pos.x > balls[0].p.x)
736         predict_pos.x = balls[0].p.x;
737
738     if (predict_pos.y > balls[0].p.y)
739         predict_pos.y = balls[0].p.y;
740
741     if (predict_pos.x < 0)
742         predict_pos.x = 0;
743
744     if (predict_pos.y < 0)
745         predict_pos.y = 0;
746
747     if (predict_pos.x > table.size.x)
748         predict_pos.x = table.size.x;
749
750     if (predict_pos.y > table.size.y)
751         predict_pos.y = table.size.y;
752
753     if (predict_pos.x < balls[0].p.x)
754         predict_pos.x = balls[0].p.x;
755
756     if (predict_pos.y < balls[0].p.y)
757         predict_pos.y = balls[0].p.y;
758
759     if (predict_pos.x > balls[0].p.x)
760         predict_pos.x = balls[0].p.x;
761
762     if (predict_pos.y > balls[0].p.y)
763         predict_pos.y = balls[0].p.y;
764
765     if (predict_pos.x < 0)
766         predict_pos.x = 0;
767
768     if (predict_pos.y < 0)
769         predict_pos.y = 0;
770
771     if (predict_pos.x > table.size.x)
772         predict_pos.x = table.size.x;
773
774     if (predict_pos.y > table.size.y)
775         predict_pos.y = table.size.y;
776
777     if (predict_pos.x < balls[0].p.x)
778         predict_pos.x = balls[0].p.x;
779
780     if (predict_pos.y < balls[0].p.y)
781         predict_pos.y = balls[0].p.y;
782
783     if (predict_pos.x > balls[0].p.x)
784         predict_pos.x = balls[0].p.x;
785
786     if (predict_pos.y > balls[0].p.y)
787         predict_pos.y = balls[0].p.y;
788
789     if (predict_pos.x < 0)
790         predict_pos.x = 0;
791
792     if (predict_pos.y < 0)
793         predict_pos.y = 0;
794
795     if (predict_pos.x > table.size.x)
796         predict_pos.x = table.size.x;
797
798     if (predict_pos.y > table.size.y)
799         predict_pos.y = table.size.y;
800
801     if (predict_pos.x < balls[0].p.x)
802         predict_pos.x = balls[0].p.x;
803
804     if (predict_pos.y < balls[0].p.y)
805         predict_pos.y = balls[0].p.y;
806
807     if (predict_pos.x > balls[0].p.x)
808         predict_pos.x = balls[0].p.x;
809
810     if (predict_pos.y > balls[0].p.y)
811         predict_pos.y = balls[0].p.y;
812
813     if (predict_pos.x < 0)
814         predict_pos.x = 0;
815
816     if (predict_pos.y < 0)
817         predict_pos.y = 0;
818
819     if (predict_pos.x > table.size.x)
820         predict_pos.x = table.size.x;
821
822     if (predict_pos.y > table.size.y)
823         predict_pos.y = table.size.y;
824
825     if (predict_pos.x < balls[0].p.x)
826         predict_pos.x = balls[0].p.x;
827
828     if (predict_pos.y < balls[0].p.y)
829         predict_pos.y = balls[0].p.y;
830
831     if (predict_pos.x > balls[0].p.x)
832         predict_pos.x = balls[0].p.x;
833
834     if (predict_pos.y > balls[0].p.y)
835         predict_pos.y = balls[0].p.y;
836
837     if (predict_pos.x < 0)
838         predict_pos.x = 0;
839
840     if (predict_pos.y < 0)
841         predict_pos.y = 0;
842
843     if (predict_pos.x > table.size.x)
844         predict_pos.x = table.size.x;
845
846     if (predict_pos.y > table.size.y)
847         predict_pos.y = table.size.y;
848
849     if (predict_pos.x < balls[0].p.x)
850         predict_pos.x = balls[0].p.x;
851
852     if (predict_pos.y < balls[0].p.y)
853         predict_pos.y = balls[0].p.y;
854
855     if (predict_pos.x > balls[0].p.x)
856         predict_pos.x = balls[0].p.x;
857
858     if (predict_pos.y > balls[0].p.y)
859         predict_pos.y = balls[0].p.y;
860
861     if (predict_pos.x < 0)
862         predict_pos.x = 0;
863
864     if (predict_pos.y < 0)
865         predict_pos.y = 0;
866
867     if (predict_pos.x > table.size.x)
868         predict_pos.x = table.size.x;
869
870     if (predict_pos.y > table.size.y)
871         predict_pos.y = table.size.y;
872
873     if (predict_pos.x < balls[0].p.x)
874         predict_pos.x = balls[0].p.x;
875
876     if (predict_pos.y < balls[0].p.y)
877         predict_pos.y = balls[0].p.y;
878
879     if (predict_pos.x > balls[0].p.x)
880         predict_pos.x = balls[0].p.x;
881
882     if (predict_pos.y > balls[0].p.y)
883         predict_pos.y = balls[0].p.y;
884
885     if (predict_pos.x < 0)
886         predict_pos.x = 0;
887
888     if (predict_pos.y < 0)
889         predict_pos.y = 0;
890
891     if (predict_pos.x > table.size.x)
892         predict_pos.x = table.size.x;
893
894     if (predict_pos.y > table.size.y)
895         predict_pos.y = table.size.y;
896
897     if (predict_pos.x < balls[0].p.x)
898         predict_pos.x = balls[0].p.x;
899
900     if (predict_pos.y < balls[0].p.y)
901         predict_pos.y = balls[0].p.y;
902
903     if (predict_pos.x > balls[0].p.x)
904         predict_pos.x = balls[0].p.x;
905
906     if (predict_pos.y > balls[0].p.y)
907         predict_pos.y = balls[0].p.y;
908
909     if (predict_pos.x < 0)
910         predict_pos.x = 0;
911
912     if (predict_pos.y < 0)
913         predict_pos.y = 0;
914
915     if (predict_pos.x > table.size.x)
916         predict_pos.x = table.size.x;
917
918     if (predict_pos.y > table.size.y)
919         predict_pos.y = table.size.y;
920
921     if (predict_pos.x < balls[0].p.x)
922         predict_pos.x = balls[0].p.x;
923
924     if (predict_pos.y < balls[0].p.y)
925         predict_pos.y = balls[0].p.y;
926
927     if (predict_pos.x > balls[0].p.x)
928         predict_pos.x = balls[0].p.x;
929
930     if (predict_pos.y > balls[0].p.y)
931         predict_pos.y = balls[0].p.y;
932
933     if (predict_pos.x < 0)
934         predict_pos.x = 0;
935
936     if (predict_pos.y < 0)
937         predict_pos.y = 0;
938
939     if (predict_pos.x > table.size.x)
940         predict_pos.x = table.size.x;
941
942     if (predict_pos.y > table.size.y)
943         predict_pos.y = table.size.y;
944
945     if (predict_pos.x < balls[0].p.x)
946         predict_pos.x = balls[0].p.x;
947
948     if (predict_pos.y < balls[0].p.y)
949         predict_pos.y = balls[0].p.y;
950
951     if (predict_pos.x > balls[0].p.x)
952         predict_pos.x = balls[0].p.x;
953
954     if (predict_pos.y > balls[0].p.y)
955         predict_pos.y = balls[0].p.y;
956
957     if (predict_pos.x < 0)
958         predict_pos.x = 0;
959
960     if (predict_pos.y < 0)
961         predict_pos.y = 0;
962
963     if (predict_pos.x > table.size.x)
964         predict_pos.x = table.size.x;
965
966     if (predict_pos.y > table.size.y)
967         predict_pos.y = table.size.y;
968
969     if (predict_pos.x < balls[0].p.x)
970         predict_pos.x = balls[0].p.x;
971
972     if (predict_pos.y < balls[0].p.y)
973         predict_pos.y = balls[0].p.y;
974
975     if (predict_pos.x > balls[0].p.x)
976         predict_pos.x = balls[0].p.x;
977
978     if (predict_pos.y > balls[0].p.y)
979         predict_pos.y = balls[0].p.y;
980
981     if (predict_pos.x < 0)
982         predict_pos.x = 0;
983
984     if (predict_pos.y < 0)
985         predict_pos.y = 0;
986
987     if (predict_pos.x > table.size.x)
988         predict_pos.x = table.size.x;
989
990     if (predict_pos.y > table.size.y)
991         predict_pos.y = table.size.y;
992
993     if (predict_pos.x < balls[0].p.x)
994         predict_pos.x = balls[0].p.x;
995
996     if (predict_pos.y < balls[0].p.y)
997         predict_pos.y = balls[0].p.y;
998
999     if (predict_pos.x > balls[0].p.x)
1000        predict_pos.x = balls[0].p.x;
1001
1002     if (predict_pos.y > balls[0].p.y)
1003        predict_pos.y = balls[0].p.y;
1004
1005     if (predict_pos.x < 0)
1006        predict_pos.x = 0;
1007
1008     if (predict_pos.y < 0)
1009        predict_pos.y = 0;
1010
1011     if (predict_pos.x > table.size.x)
1012        predict_pos.x = table.size.x;
1013
1014     if (predict_pos.y > table.size.y)
1015        predict_pos.y = table.size.y;
1016
1017     if (predict_pos.x < balls[0].p.x)
1018        predict_pos.x = balls[0].p.x;
1019
1020     if (predict_pos.y < balls[0].p.y)
1021        predict_pos.y = balls[0].p.y;
1022
1023     if (predict_pos.x > balls[0].p.x)
1024        predict_pos.x = balls[0].p.x;
1025
1026     if (predict_pos.y > balls[0].p.y)
1027        predict_pos.y = balls[0].p.y;
1028
1029     if (predict_pos.x < 0)
1030        predict_pos.x = 0;
1031
1032     if (predict_pos.y < 0)
1033        predict_pos.y = 0;
1034
1035     if (predict_pos.x > table.size.x)
1036        predict_pos.x = table.size.x;
1037
1038     if (predict_pos.y > table.size.y)
1039        predict_pos.y = table.size.y;
1040
1041     if (predict_pos.x < balls[0].p.x)
1042        predict_pos.x = balls[0].p.x;
1043
1044     if (predict_pos.y < balls[0].p.y)
1045        predict_pos.y = balls[0].p.y;
1046
1047     if (predict_pos.x > balls[0].p.x)
1048        predict_pos.x = balls[0].p.x;
1049
1050     if (predict_pos.y > balls[0].p.y)
1051        predict_pos.y = balls[0].p.y;
1052
1053     if (predict_pos.x < 0)
1054        predict_pos.x = 0;
1055
1056     if (predict_pos.y < 0)
1057        predict_pos.y = 0;
1058
1059     if (predict_pos.x > table.size.x)
1060        predict_pos.x = table.size.x;
1061
1062     if (predict_pos.y > table.size.y)
1063        predict_pos.y = table.size.y;
1064
1065     if (predict_pos.x < balls[0].p.x)
1066        predict_pos.x = balls[0].p.x;
1067
1068     if (predict_pos.y < balls[0].p.y)
1069        predict_pos.y = balls[0].p.y;
1070
1071     if (predict_pos.x > balls[0].p.x)
1072        predict_pos.x = balls[0].p.x;
1073
1074     if (predict_pos.y > balls[0].p.y)
1075        predict_pos.y = balls[0].p.y;
1076
1077     if (predict_pos.x < 0)
1078        predict_pos.x = 0;
1079
1080     if (predict_pos.y < 0)
1081        predict_pos.y = 0;
1082
1083     if (predict_pos.x > table.size.x)
1084        predict_pos.x = table.size.x;
1085
1086     if (predict_pos.y > table.size.y)
1087        predict_pos.y = table.size.y;
1088
1089     if (predict_pos.x < balls[0].p.x)
1090        predict_pos.x = balls[0].p.x;
1091
1092     if (predict_pos.y < balls[0].p.y)
1093        predict_pos.y = balls[0].p.y;
1094
1095     if (predict_pos.x > balls[0].p.x)
1096        predict_pos.x = balls[0].p.x;
1097
1098     if (predict_pos.y > balls[0].p.y)
1099        predict_pos.y = balls[0].p.y;
1100
1101     if (predict_pos.x < 0)
1102        predict_pos.x = 0;
1103
1104     if (predict_pos.y < 0)
1105        predict_pos.y = 0;
1106
1107     if (predict_pos.x > table.size.x)
1108        predict_pos.x = table.size.x;
1109
1110     if (predict_pos.y > table.size.y)
1111        predict_pos.y = table.size.y;
1112
1113     if (predict_pos.x < balls[0].p.x)
1114        predict_pos.x = balls[0].p.x;
1115
1116     if (predict_pos.y < balls[0].p.y)
1117        predict_pos.y = balls[0].p.y;
1118
1119     if (predict_pos.x > balls[0].p.x)
1120        predict_pos.x = balls[0].p.x;
1121
1122     if (predict_pos.y > balls[0].p.y)
1123        predict_pos.y = balls[0].p.y;
1124
1125     if (predict_pos.x < 0)
1126        predict_pos.x = 0;
1127
1128     if (predict_pos.y < 0)
1129        predict_pos.y = 0;
1130
1131     if (predict_pos.x > table.size.x)
1132        predict_pos.x = table.size.x;
1133
1134     if (predict_pos.y > table.size.y)
1135        predict_pos.y = table.size.y;
1136
1137     if (predict_pos.x < balls[0].p.x)
1138        predict_pos.x = balls[0].p.x;
1139
1140     if (predict_pos.y < balls[0].p.y)
1141        predict_pos.y = balls[0].p.y;
1142
1143     if (predict_pos.x > balls[0].p.x)
1144        predict_pos.x = balls[0].p.x;
1145
1146     if (predict_pos.y > balls[0].p.y)
1147        predict_pos.y = balls[0].p.y;
1148
1149     if (predict_pos.x < 0)
1150        predict_pos.x = 0;
1151
1152     if (predict_pos.y < 0)
1153        predict_pos.y = 0;
1154
1155     if (predict_pos.x > table.size.x)
1156        predict_pos.x = table.size.x;
1157
1158     if (predict_pos.y > table.size.y)
1159        predict_pos.y = table.size.y;
1160
1161     if (predict_pos.x < balls[0].p.x)
1162        predict_pos.x = balls[0].p.x;
1163
1164     if (predict_pos.y < balls[0].p.y)
1165        predict_pos.y = balls[0].p.y;
1166
1167     if (predict_pos.x > balls[0].p.x)
1168        predict_pos.x = balls[0].p.x;
1169
1170     if (predict_pos.y > balls[0].p.y)
1171        predict_pos.y = balls[0].p.y;
1172
1173     if (predict_pos.x < 0)
1174        predict_pos.x = 0;
1175
1176     if (predict_pos.y < 0)
1177        predict_pos.y = 0;
1178
1179     if (predict_pos.x > table.size.x)
1180        predict_pos.x = table.size.x;
1181
1182     if (predict_pos.y > table.size.y)
1183        predict_pos.y = table.size.y;
1184
1185     if (predict_pos.x < balls[0].p.x)
1186        predict_pos.x = balls[0].p.x;
1187
1188     if (predict_pos.y < balls[0].p.y)
1189        predict_pos.y = balls[0].p.y;
1190
1191     if (predict_pos.x > balls[0].p.x)
1192        predict_pos.x = balls[0].p.x;
1193
1194     if (predict_pos.y > balls[0].p.y)
1195        predict_pos.y = balls[0].p.y;
1196
1197     if (predict_pos.x < 0)
1198        predict_pos.x = 0;
1199
1200     if (predict_pos.y < 0)
1201        predict_pos.y = 0;
1202
1203     if (predict_pos.x > table.size.x)
1204        predict_pos.x = table.size.x;
1205
1206     if (predict_pos.y > table.size.y)
1207        predict_pos.y = table.size.y;
1208
1209     if (predict_pos.x < balls[0].p.x)
1210        predict_pos.x = balls[0].p.x;
1211
1212     if (predict_pos.y < balls[0].p.y)
1213        predict_pos.y = balls[0].p.y;
1214
1215     if (predict_pos.x > balls[0].p.x)
1216        predict_pos.x = balls[0].p.x;
1217
1218     if (predict_pos.y > balls[0].p.y)
1219        predict_pos.y = balls[0].p.y;
1220
1221     if (predict_pos.x < 0)
1222        predict_pos.x = 0;
1223
1224     if (predict_pos.y < 0)
1225        predict_pos.y = 0;
1226
1227     if (predict_pos.x > table.size.x)
1228        predict_pos.x = table.size.x;
1229
1230     if (predict_pos.y > table.size.y)
1231        predict_pos.y = table.size.y;
1232
1233     if (predict_pos.x < balls[0].p.x)
1234        predict_pos.x = balls[0].p.x;
1235
1236     if (predict_pos.y < balls[0].p.y)
1237        predict_pos.y = balls[0].p.y;
1238
1239     if (predict_pos.x > balls[0].p.x)
1240        predict_pos.x = balls[0].p.x;
1241
1242     if (predict_pos.y > balls[0].p.y)
1243        predict_pos.y = balls[0].p.y;
1244
1245     if (predict_pos.x < 0)
1246        predict_pos.x = 0;
1247
1248     if (predict_pos.y < 0)
1249        predict_pos.y = 0;
1250
1251     if (predict_pos.x > table.size.x)
1252        predict_pos.x = table.size.x;
1253
1254     if (predict_pos.y > table.size.y)
1255        predict_pos.y = table.size.y;
1256
1257     if (predict_pos.x < balls[0].p.x)
1258        predict_pos.x = balls[0].p.x;
1259
1260     if (predict_pos.y < balls[0].p.y)
1261        predict_pos.y = balls[0].p.y;
1262
1263     if (predict_pos.x > balls[0].p.x)
1264        predict_pos.x = balls[0].p.x;
1265
1266     if (predict_pos.y > balls[0].p.y)
1267        predict_pos.y = balls[0].p.y;
1268
1269     if (predict_pos.x < 0)
1270        predict_pos.x = 0;
1271
1272     if (predict_pos.y < 0)
1273        predict_pos.y = 0;
1274
1275     if (predict_pos.x > table.size.x)
1276        predict_pos.x = table.size.x;
1277
1278     if (predict_pos.y > table.size.y)
1279        predict_pos.y = table.size.y;
1280
1281     if (predict_pos.x < balls[0].p.x)
1282        predict_pos.x = balls[0].p.x;
1283
1284     if (predict_pos.y < balls[0].p.y)
1285        predict_pos.y = balls[0].p.y;
1286
1287     if (predict_pos.x > balls[0].p.x)
1288        predict_pos.x = balls[0].p.x;
1289
1290     if (predict_pos.y > balls[0].p.y)
1291        predict_pos.y = balls[0].p.y;
1292
1293     if (predict_pos.x < 0)
1294        predict_pos.x = 0;
1295
1296     if (predict_pos.y < 0)
1297        predict_pos.y = 0;
1298
1299     if (predict_pos.x > table.size.x)
1300        predict_pos.x = table.size.x;
1301
1302     if (predict_pos.y > table.size.y)
1303        predict_pos.y = table.size.y;
1304
1305     if (predict_pos.x < balls[0].p.x)
1306        predict_pos.x = balls[0].p.x;
1307
1308     if (predict_pos.y < balls[0].p.y)
1309        predict_pos.y = balls[0].p.y;
1310
1311     if (predict_pos.x > balls[0].p.x)
1312        predict_pos.x = balls[0].p.x;
1313
1314     if (predict_pos.y > balls[0].p.y)
1315        predict_pos.y = balls[0].p.y;
1316
1317     if (predict_pos.x < 0)
1318        predict_pos.x = 0;
1319
1320     if (predict_pos.y < 0)
1321        predict_pos.y = 0;
1322
1323     if (predict_pos.x > table.size.x)
1324        predict_pos.x = table.size.x;
1325
1326    
```

```

455         if (predict_dist < predict_min) {
456             target = i;
457             predict_min = predict_dist;
458         }
459     }
460 }
461
462 // 予測線を描画
463 glDisable(GL_LIGHTING);
464 glPushMatrix();
465 glTranslated(balls[0].p.x, balls[0].p.y, 0);
466 glRotated(degree(cue.angle), 0, 0, 1);
467 glBegin(GL_LINES);
468 glColor3d(0.0, 0.0, 0.0);
469 glVertex2d(0, 0);
470 glVertex2d(predict_min - balls[0].r, 0);
471 glEnd();
472
473 drawCircle(predict_min, 0, balls[0].r, 32);
474 glPopMatrix();
475 glEnable(GL_LIGHTING);
476 }
477
478 // 球
479 for (i = BALL_NUM - 1; i >= 0; i--) {
480     if (balls[i].exist)
481         drawBall(balls[i]);
482 }
483
484 // 無効マーク
485 if (status == Put && !canPut())
486     putSprite(invalid_image, balls[0].p.x, balls[0].p.y, balls[0].r * 2, balls[0].r
* 2, 1);
487
488 if ((status == Stop || status == Pull) && target && balls[target].num != next)
489     putSprite(invalid_image, balls[target].p.x, balls[target].p.y, balls[target].r
* 2, balls[target].r * 2, 1);
490
491 // キュー
492 glPushMatrix();
493 glTranslated(cue.p.x, cue.p.y, 0);
494 glRotated(degree(cue.angle) + 180, 0, 0, 1);
495 putSprite(cue.image, balls[0].r + CUE_W / 2 + cue.power * 2, 0, CUE_W, CUE_H, 1);
496 glPopMatrix();
497
498 if (status == Result) {
499     // 結果表示
500     if (win_left > 0) {
501         double step = 1 - 2.0 * win_left / RESULT_TIME;
502         putSprite(win_images[turn], 0, 0, ASPECT, 0.5, 1 - pow(step, 4));
503     }
504 } else {
505     // ターン表示
506     if (turn_left > 0) {
507         double step = 1 - 2.0 * turn_left / TURN_TIME;
508         putSprite(turn_images[turn], 0, 0, ASPECT, 0.5, 1 - pow(step, 4));

```

```

509         }
510     }
511 }
512
513     glFlush();
514 }
515
516 // ウィンドウサイズ変更
517 void Reshape(int w, int h) {
518     // ウィンドウの縦横比
519     double ratio = (double)w / h;
520
521     // 座標系再設定
522     glViewport(0, 0, w, h);
523     glLoadIdentity();
524
525     if (ratio > ASPECT)
526         gluOrtho2D(-ratio, ratio, -1, 1);
527     else
528         gluOrtho2D(-ASPECT, ASPECT, -ASPECT / ratio, ASPECT / ratio);
529 }
530
531 // タイマー
532 void Timer(int value) {
533     // 次のタイマーを登録
534     glutTimerFunc(1000.0 / FPS + 0.5, Timer, 0);
535
536     if (scene == Game)
537         update();
538
539     glutPostRedisplay();
540 }
541
542 // マウスクリック
543 void Mouse(int button, int stat, int x, int y) {
544     int i;
545
546     mouse = convertPoint(x, y);
547
548     if (button == GLUT_LEFT_BUTTON) {
549         if (scene == Title) {
550             if (stat == GLUT_UP) {
551                 // 人vs人ボタンクリック
552                 if (fabs(mouse.x) <= ASPECT / 2 && -0.25 <= mouse.y && mouse.y <= 0.25) {
553                     scene = Game;
554                     turn = Player1;
555                 }
556
557                 // 人vsCPUボタンクリック
558                 if (fabs(mouse.x) <= ASPECT / 2 && -0.875 <= mouse.y && mouse.y <= -0.375)
559                 {
560                     scene = Game;
561                     turn = Player;
562                 }
563             }
564         }
565     }

```

```

564         if (scene == Game) {
565             status = Stop;
566             break_shot = 1;
567             next = 1;
568             turn_left = TURN_TIME;
569             init();
570         }
571     }
572 } else {
573     // CPU操作時はクリック無効
574     if (turn == CPU)
575         return;
576
577     switch (status) {
578         // 静止中
579         case Stop: {
580             if (stat == GLUT_DOWN) {
581                 status = Pull;
582                 cue.angle = angle(sub(mouse, balls[0].p));
583             }
584
585             break;
586         }
587
588         // キューを引いている
589         case Pull: {
590             if (stat == GLUT_UP) {
591                 status = Move;
592
593                 // 状態を保持
594                 for (i = 0; i < BALL_NUM; i++)
595                     prev_balls[i] = balls[i];
596
597                 // 手球に初速度を与える
598                 balls[0].v = mult(vector(cos(cue.angle), sin(cue.angle)), cue.power
599 );
599                 cue.power = 0;
600                 first_touch = 0;
601             }
602
603             break;
604         }
605
606         // 手球配置中
607         case Put: {
608             if (stat == GLUT_DOWN && canPut()) {
609                 status = Stop;
610                 cue.p = balls[0].p = mouse;
611             }
612
613             break;
614         }
615
616         default:
617             break;
618     }

```

```

619     }
620   }
621 }
622
623 // マウス移動
624 void PassiveMotion(int x, int y) {
625   mouse = convertPoint(x, y);
626 }
```

リスト 3 table.h

```

1 #ifndef TABLE_H
2 #define TABLE_H
3
4 #include <GL/glut.h>
5 #include <GL/glpng.h>
6
7 #include "ball.h"
8 #include "vector.h"
9
10 // 台
11 struct table {
12   struct vector size; // 大きさ
13   double pocket_r; // ポケットの半径
14   double wall_loss; // 壁衝突時の速度損失
15   GLuint image; // 画像
16 };
17
18 void reflectTable(struct table, struct ball *);
19 int pocketTouching(struct table, struct ball);
20 void pocketIn(struct table, struct ball *);
21
22 #endif
```

リスト 4 table.c

```

1 #include <math.h>
2
3 #include "table.h"
4 #include "ball.h"
5 #include "vector.h"
6
7 // 台との反射
8 void reflectTable(struct table table, struct ball *ball) {
9   // 右側
10  if (ball->p.x > table.size.x - ball->r) {
11    ball->p.x = table.size.x - ball->r;
12    ball->p.y = ball->p.y - tan(angle(ball->v)) * (ball->p.x - table.size.x + ball->r);
13    ball->v.x *= -1;
14    ball->v = mult(ball->v, table.wall_loss);
15  }
16
17  // 左側
18  if (ball->p.x < -table.size.x + ball->r) {
19    ball->p.x = -table.size.x + ball->r;
20    ball->p.y = ball->p.y - tan(angle(ball->v)) * (ball->p.x + table.size.x - ball->r);
```

```

21     ball->v.x *= -1;
22     ball->v = mult(ball->v, table.wall_loss);
23 }
24
25 // 上側
26 if (ball->p.y > table.size.y - ball->r) {
27     ball->p.x = ball->p.x - (ball->p.y - table.size.y + ball->r) / tan(angle(ball->v));
28     ball->p.y = table.size.y - ball->r;
29     ball->v.y *= -1;
30     ball->v = mult(ball->v, table.wall_loss);
31 }
32
33 // 下側
34 if (ball->p.y < -table.size.y + ball->r) {
35     ball->p.x = ball->p.x - (ball->p.y + table.size.y - ball->r) / tan(angle(ball->v));
36     ball->p.y = -table.size.y + ball->r;
37     ball->v.y *= -1;
38     ball->v = mult(ball->v, table.wall_loss);
39 }
40 }
41
42 // ポケットに入るか
43 int pocketTouching(struct table table, struct ball ball) {
44     int i;
45
46     // ポケットの座標
47     struct vector pockets[6] = {
48         {-table.size.x, table.size.y},
49         {0, table.size.y},
50         {table.size.x, table.size.y},
51         {-table.size.x, -table.size.y},
52         {0, -table.size.y},
53         {table.size.x, -table.size.y}
54     };
55
56     for (i = 0; i < 6; i++) {
57         if (dist(ball.p, pockets[i]) < table.pocket_r)
58             return 1;
59     }
60
61     return 0;
62 }
63
64 // ポケットに入る
65 void pocketIn(struct table table, struct ball *ball) {
66     if (pocketTouching(table, *ball)) {
67         ball->exist = 0;
68         ball->v = ZERO;
69     }
70 }
```

リスト 5 ball.h

```

1 #ifndef BALL_H
2 #define BALL_H
3
```

```

4 #include <GL/glut.h>
5 #include <GL/glpng.h>
6
7 #include "vector.h"
8
9 #define FRICTION 0.0002 // 摩擦
10#define BALL_LOSS 0.95 // 球衝突時の速度損失
11
12// 球
13struct ball {
14    int exist;           // 存在しているか
15    int num;            // 番号 (0:白玉)
16    double r;           // 半径
17    struct vector p;   // 位置
18    struct vector v;   // 速度
19    struct vector dir; // 進行方向
20    double angle;      // 角度
21    GLuint image;       // 画像
22};
23
24void initBall(struct ball *, int, struct vector, double);
25void moveBall(struct ball *);
26void drawBall(struct ball);
27int ballColliding(struct ball, struct ball);
28void reflectBall(struct ball *, struct ball *, int);
29
30#endif

```

リスト 6 ball.c

```

1 #include <math.h>
2
3 #include <GL/glut.h>
4 #include <GL/glpng.h>
5
6 #include "ball.h"
7 #include "vector.h"
8
9 // 角度を変換
10#define radian(deg) (rad * M_PI / 180.0)
11#define degree(rad) (rad * 180.0 / M_PI)
12
13// ball構造体を初期化
14void initBall(struct ball *ball, int num, struct vector p, double r) {
15    ball->num = num;
16    ball->exist = 1;
17    ball->r = r;
18    ball->p = p;
19    ball->v = ZERO;
20    ball->dir = vector(1, 0);
21    ball->angle = 0;
22}
23
24// 移動
25void moveBall(struct ball *ball) {
26    ball->p = add(ball->p, ball->v);

```

```

27     ball->angle = fmod(ball->angle + mag(ball->v) / ball->r, 2 * M_PI);
28     if (!isZero(ball->v)) ball->dir = normal(ball->v);
29
30     // 摩擦
31     if (mag(ball->v) <= 0.001)
32         ball->v = ZERO;
33     else
34         ball->v = mult(ball->v, 1 - FRICTION / mag(ball->v));
35 }
36
37 // ボールを描画
38 void drawBall(struct ball ball) {
39     // 球体の設定
40     GLUquadricObj* sphere = gluNewQuadric();
41     gluQuadricDrawStyle(sphere, GLU_FILL);
42     gluQuadricNormals(sphere, GLU_SMOOTH);
43     gluQuadricTexture(sphere, GL_TRUE);
44
45     glEnable(GL_TEXTURE_2D);
46     glBindTexture(GL_TEXTURE_2D, ball.image);
47
48     glPushMatrix();
49     glTranslated(ball.p.x, ball.p.y, ball.r);
50     glRotated(90, 0, 1, 0);
51     glRotated(90, 1, 0, 0);
52     glRotated(degree(ball.angle), 0, -ball.dir.y, -ball.dir.x);
53     gluSphere(sphere, ball.r, 32, 32);
54     glPopMatrix();
55
56     glDisable(GL_TEXTURE_2D);
57 }
58
59 // ボールが衝突しているか
60 int ballColliding(struct ball ballA, struct ball ballB) {
61     return dist(ballA.p, ballB.p) < ballA.r + ballB.r;
62 }
63
64 // ボール同士の反射
65 void reflectBall(struct ball *ballA, struct ball *ballB, int break_shot) {
66     if (ballColliding(*ballA, *ballB)) {
67         struct vector p_ab, v_ab, temp;
68         double a, b, c, D, t;
69
70         // ボールの重なりを修正
71         p_ab = sub(ballA->p, ballB->p);
72         v_ab = sub(ballA->v, ballB->v);
73         a = pow(mag(v_ab), 2);
74         b = inner(p_ab, v_ab);
75         c = pow(mag(p_ab), 2) - pow(ballA->r + ballB->r, 2);
76         D = pow(b, 2) - a * c;
77
78         if (D > 0 && !break_shot) {
79             t = (-b - sqrt(D)) / a;
80             ballA->p = add(ballA->p, mult(ballA->v, t));
81             ballB->p = add(ballB->p, mult(ballB->v, t));
82         } else {

```

```

83         temp = mult(normal(p_ab), (ballA->r + ballB->r - mag(p_ab)) / 2);
84         ballA->p = add(ballA->p, temp);
85         ballB->p = sub(ballB->p, temp);
86     }
87
88     // 衝突後の速度を決定
89     p_ab = sub(ballA->p, ballB->p);
90     v_ab = sub(ballA->v, ballB->v);
91     temp = mult(normal(p_ab), -inner(normal(p_ab), v_ab));
92     ballA->v = mult(add(ballA->v, temp), BALL_LOSS);
93     ballB->v = mult(sub(ballB->v, temp), BALL_LOSS);
94 }
95 }
```

リスト 7 vector.h

```

1 #ifndef VECTOR_H
2 #define VECTOR_H
3
4 // 二次元ベクトル
5 struct vector {
6     double x;
7     double y;
8 };
9
10 struct vector vector(double, double);
11
12 struct vector add(struct vector, struct vector);
13 struct vector sub(struct vector, struct vector);
14 struct vector mult(struct vector, double);
15 struct vector divi(struct vector, double);
16 struct vector rotate(struct vector, double);
17 struct vector normal(struct vector);
18
19 double mag(struct vector);
20 double angle(struct vector);
21 double angle2(struct vector, struct vector);
22 double dist(struct vector, struct vector);
23 double inner(struct vector, struct vector);
24 int isZero(struct vector);
25
26 extern const struct vector ZERO;
27
28 #endif
```

リスト 8 vector.c

```

1 #include <stdio.h>
2 #include <math.h>
3 #include <float.h>
4
5 #include "vector.h"
6
7 // ゼロベクトル
8 const struct vector ZERO = {0, 0};
9
```

```

10 // ベクトルを生成
11 struct vector vector(double x, double y) {
12     struct vector a = {x, y};
13     return a;
14 }
15
16 // 加算
17 struct vector add(struct vector a, const struct vector b) {
18     return vector(a.x + b.x, a.y + b.y);
19 }
20
21 // 減算
22 struct vector sub(struct vector a, const struct vector b) {
23     return vector(a.x - b.x, a.y - b.y);
24 }
25
26 // 乗算
27 struct vector mult(struct vector a, double r) {
28     return vector(a.x * r, a.y * r);
29 }
30
31 // 除算
32 struct vector divi(struct vector a, double r) {
33     return vector(a.x / r, a.y / r);
34 }
35
36 // 回転
37 struct vector rotate(struct vector a, double r) {
38     return vector(a.x * cos(r) - a.y * sin(r), a.x * sin(r) + a.y * cos(r));
39 }
40
41 // 正規化
42 struct vector normal(struct vector a) {
43     return divi(a, mag(a));
44 }
45
46 // 大きさ
47 double mag(struct vector a) {
48     return sqrt(pow(a.x, 2) + pow(a.y, 2));
49 }
50
51 // ベクトルの角度
52 double angle(struct vector a) {
53     return atan2(a.y, a.x);
54 }
55
56 // 2つのベクトルのなす角度
57 double angle2(struct vector a, struct vector b) {
58     return acos(inner(a, b) / (mag(a) * mag(b)));
59 }
60
61 // 距離
62 double dist(struct vector a, struct vector b) {
63     return mag(sub(a, b));
64 }
65

```

```

66 // 内積
67 double inner(struct vector a, struct vector b) {
68     return a.x * b.x + a.y * b.y;
69 }
70
71 // ゼロベクトルか
72 int isZero(struct vector a) {
73     return mag(a) <= DBL_EPSILON;
74 }

```

リスト 9 shape.h

```

1 #ifndef SHAPE_H
2 #define SHAPE_H
3
4 void drawCircle(double, double, double, int);
5 void putSprite(GLuint, double, double, double, double);
6
7 #endif

```

リスト 10 shape.c

```

1 #include <math.h>
2
3 #include <GL/glut.h>
4 #include <GL/glpng.h>
5
6 #include "shape.h"
7
8 // 正円を描画
9 void drawCircle(double x, double y, double r, int h) {
10     int i;
11
12     glBegin(GL_LINE_LOOP);
13
14     for (i = 0; i < h; i++) {
15         double theta = 2 * M_PI * i / h;
16         glVertex2d(x + r * sin(theta), y + r * cos(theta));
17     }
18
19     glEnd();
20 }
21
22 // 画像を表示
23 // num : 画像の番号
24 // x, y : 座標
25 // w, h : サイズ
26 // a : 不透明度
27 void putSprite(GLuint num, double x, double y, double w, double h, double a) {
28     glDisable(GL_LIGHTING);
29     glEnable(GL_TEXTURE_2D);
30     glBindTexture(GL_TEXTURE_2D, num);
31     glColor4d(1.0, 1.0, 1.0, a);
32
33     glBegin(GL_QUADS);
34

```

```

35     glTexCoord2d(0, 0);
36     glVertex2d(x - w / 2, y + h / 2);
37
38     glTexCoord2d(0, 1);
39     glVertex2d(x - w / 2, y - h / 2);
40
41     glTexCoord2d(1, 1);
42     glVertex2d(x + w / 2, y - h / 2);
43
44     glTexCoord2d(1, 0);
45     glVertex2d(x + w / 2, y + h / 2);
46
47     glEnd();
48     glEnable(GL_LIGHTING);
49     glDisable(GL_TEXTURE_2D);
50 }
```

リスト 11 Makefile

```

1 TARGET = j16426.exe
2
3 SRCS = billiard.c ball.c table.c vector.c shape.c
4 ICON = icon.ico
5
6 ICON_RC = ${ICON:.ico=.rc}
7 ICON_OBJ = ${ICON:.ico=.o}
8 OBJS = ${SRCS:.c=.o} $(ICON_OBJ)
9
10 HEADERS = billiard.h ball.h table.h vector.h shape.h
11
12 CC = gcc
13 CCFLAGS = -Wall -I/usr/include/opengl
14 LD = gcc
15 LDFLAGS =
16 LIBS = -lm -lglpng -lglut32 -lglu32 -lopengl32
17
18 $(TARGET): $(OBJS)
19     $(LD) $(OBJS) $(LDFLAGS) -o $(TARGET) $(LIBS)
20
21 .SUFFIXES: .o .c
22 .c.o:
23     $(CC) $(CCFLAGS) -c $<
24
25 $(OBJS): $(HEADERS) Makefile
26
27 $(ICON_OBJ): $(ICON) $(ICON_RC)
28     windres -i $(ICON_RC) -o $(ICON_OBJ)
29
30 .PHONY: clean
31 clean:
32     rm -f $(TARGET) $(OBJS) core --
```

6 リソース

プログラム中で用いられる画像を図 1-21, アプリケーションのアイコンを図 27 に示す。



図 1 ball_0.png

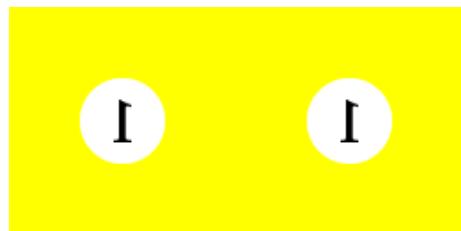


図 2 ball_1.png

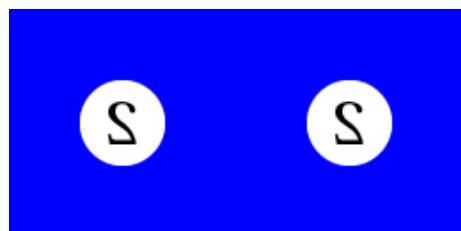


図 3 ball_2.png

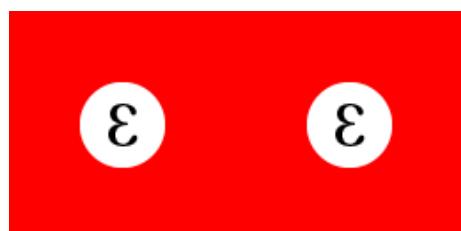


図 4 ball_3.png

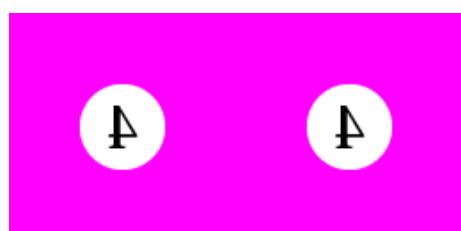


図 5 ball_4.png

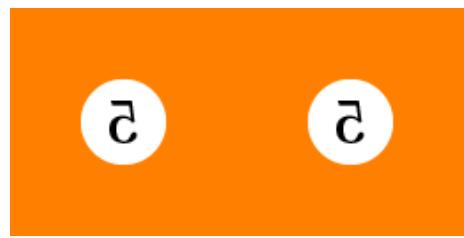


図 6 ball_5.png

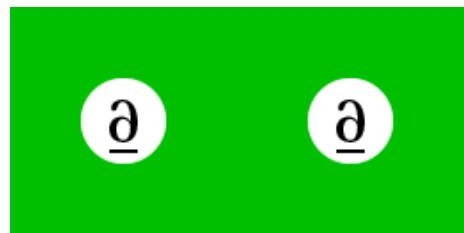


図 7 ball_6.png

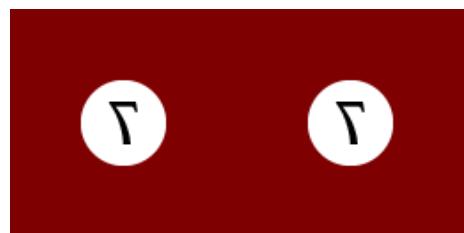


図 8 ball_7.png

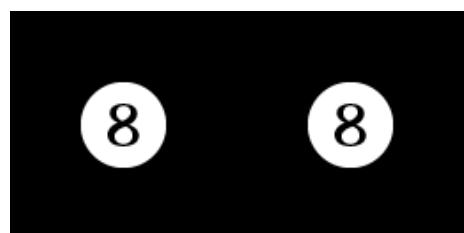


図 9 ball_8.png



図 10 ball_9.png

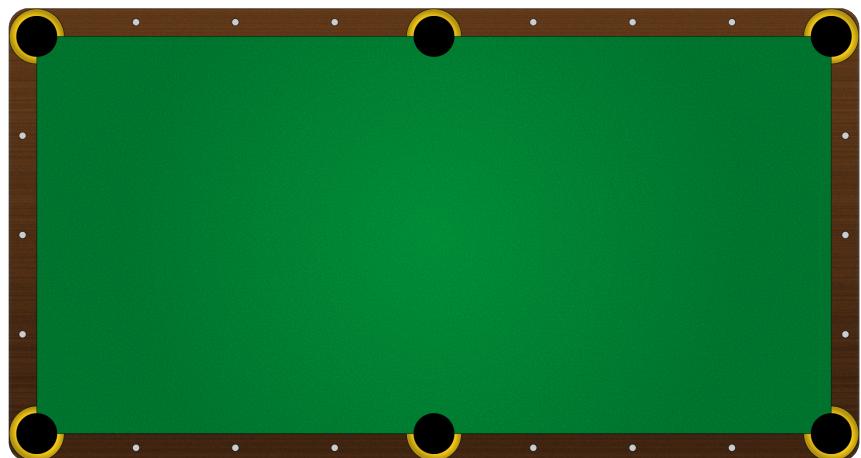


図 11 table.png



図 12 cue.png



図 13 invalid.png



図 14 turn_0.png



図 15 turn_1.png

Playerのターン

図 16 turn_2.png

CPUのターン

図 17 turn_3.png

Player1の勝利！

図 18 win_0.png

Player2の勝利！

図 19 win_1.png

Playerの勝利！

図 20 win_2.png

CPUの勝利

図 21 win_3.png

REAL BILLIARD

図 22 title.png



図 23 vshuman_0.png



図 24 vshuman_1.png



図 25 vscpu_0.png



図 26 vscpu_1.png



図 27 icon.ico

7 ビルド方法

GLUT と glpng が導入されている Cygwin 上で、次のコマンドを入力することでビルド、実行することができる。

```
$ make  
$ ./j16426.exe
```

8 実行結果

本プログラムを実行したときに表示される画面を図 28 に示す。この画面で人 vs 人か、人 vs コンピュータを選ぶと、図 29 のようなゲーム画面になる。図 29 を見ると、ビリヤード台の上に 3D の球が表示されており、回転運動もしていることが分かる。

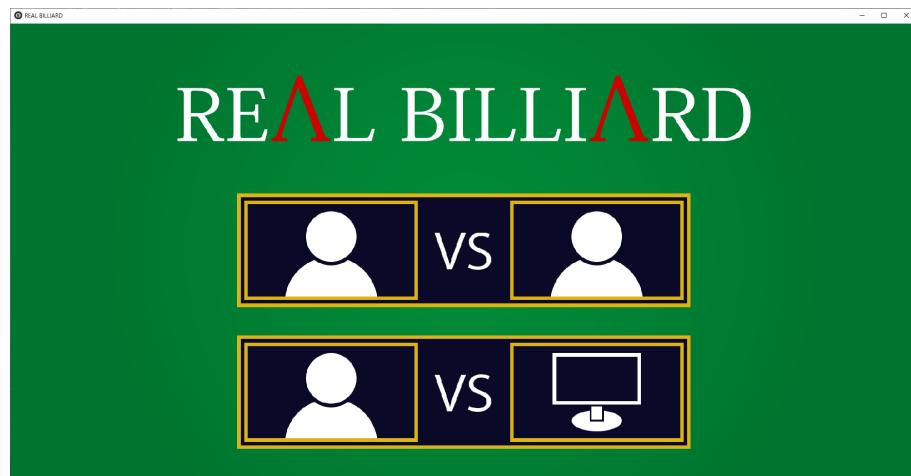


図 28 タイトル画面

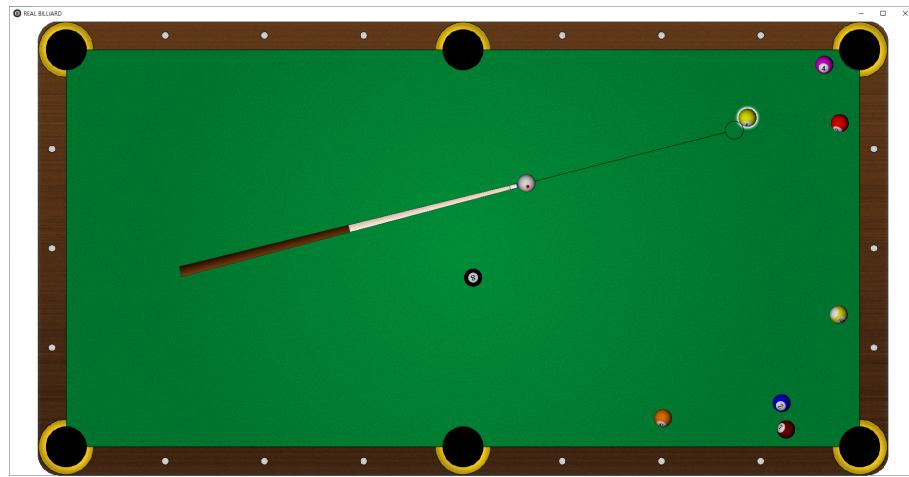


図 29 ゲーム画面

8.1 アニメーション

アニメーションは、`glutTimerFunc` 関数を用いて、短い間隔で描画関数を呼び出すことによって実現している。タイマーのコールバック関数である `Timer` の中でさらにタイマーを登録することで、連続的なアニメーションを表現することが出来る。

8.2 座標系

ウィンドウを横長になるように広げた状態を図 29、縦長になるように縮めた状態を図 30 に示す。図 29 を見ると、ウィンドウサイズに合わせて画面が拡大され、ウィンドウの中心に表示されている。図 30 を見ると、ウィンドウサイズを小さくしても見切れることなく、ウィンドウの横幅いっぱいに表示されていることが分かる。



図 30 横に長いウィンドウ

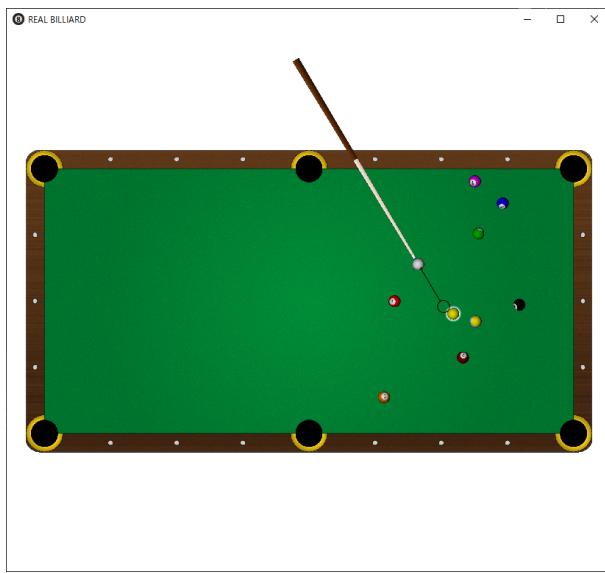


図 31 縦に長いウィンドウ

これは、ウィンドウサイズが変更された際に逐次座標系を変更しているためである。ウィンドウが横に長い際の座標系は図 32, 縦に長い際は図 33 のようになる。

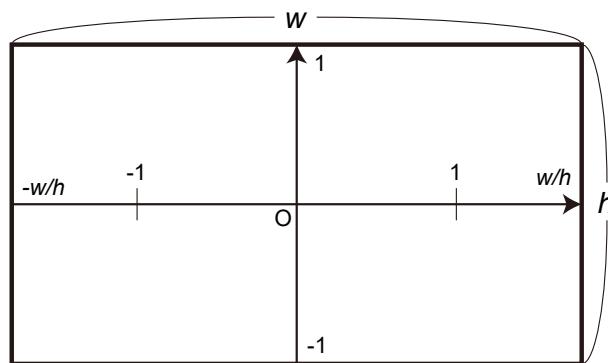


図 32 横に長い座標系

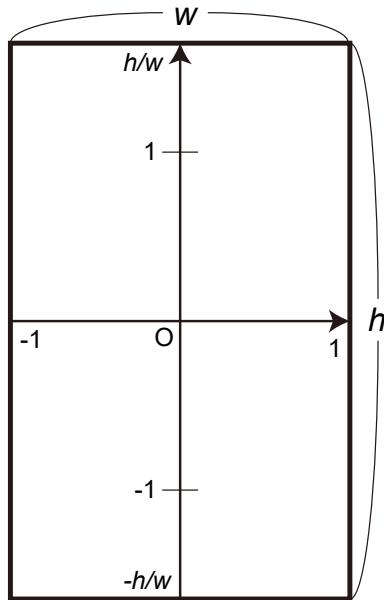


図 33 縦に長い座標系

この設定を行っているのが `Reshape` 関数である。ウィンドウサイズが変更された際に呼び出されるコールバック関数として `glutReshapeFunc` 関数で登録されている。`gluOrtho2D` という関数で画面端に対する座標を指定することができる。これを図 32, 33 のように設定することで、画面サイズに対する比で座標を指定することになるので、ウィンドウサイズが変更されてもそれに合わせて拡大縮小されるようになる。また、 x 軸方向と y 軸方向の座標の比を 1:1 になるようにしているため、横長、縦長にしても画面が潰れることがない。

8.3 球の衝突

球の衝突処理は、ball.c の `reflectBall` 関数で行っている。図 34 のように質量の等しい 2 つの球 a, b が速度 v_a, v_b で衝突した際、衝突後の速度 v'_a, v'_b は式 (1), (2) のようになる [2]。なお、 $x_{ab} = x_a - x_b$, $\hat{x} = \frac{x}{|x|}$ を表す。

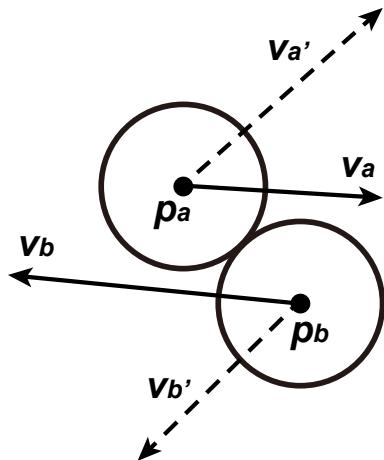


図 34 球の衝突

$$v'_a = v_a + (v_{ab} \cdot \widehat{p_{ab}}) \widehat{p_{ab}} \quad (1)$$

$$v'_b = v_b - (v_{ab} \cdot \hat{p}_{ab}) \hat{p}_{ab} \quad (2)$$

8.4 ベクトル

物理演算を行う際、ベクトルの計算が非常に多くなるため、vector.c にベクトルを表す構造体を定義した。四則演算だけでなく、2つのベクトルの成す角や内積を計算する関数を作成した。これにより、数値計算を効率的に行うことができた。

8.5 予測線

図 29 を見ると、手球の通る軌道に線が描画されている。これには、図 35 のように、速度 v_a で移動する球 a が、静止している球 b に衝突するかどうかを考える。

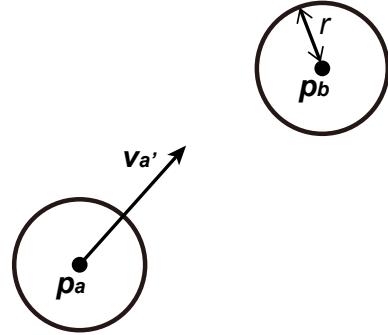


図 35 ポールの衝突予想

これは、点 p_a を通り、傾きが v_a の直線と、点 p_b の距離が $2r$ 以下であれば衝突する。点 $A(x_0, y_0)$ と直線 $ax + by + c = 0$ の距離 d は、式 (3) で与えられるため、事前に手球と衝突するかどうかを予測することができる。

$$d = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}} \quad (3)$$

8.6 コンピュータ

図 36 のように、球 a を b に当て、座標 p_p に位置するポケットに落としたい場合を考える。

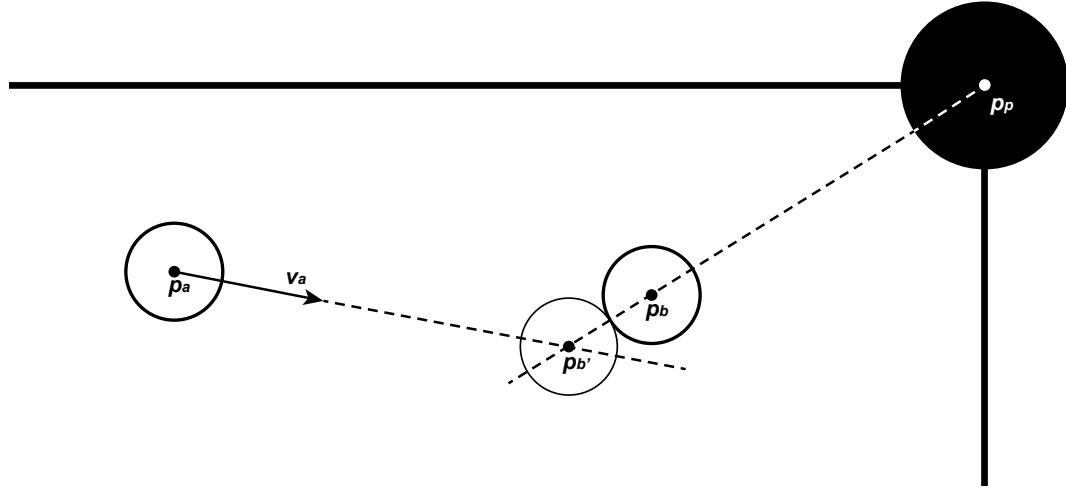


図 36 コンピュータの思考

p_b と p_p を結ぶ線上にあり、球 b と接する球の座標を p'_b とする。このとき、球 a を p'_b の方向に発射することで、球 b をポケットに落とすことができる。

ただし、ゲーム性を考えて、本プログラムのコンピュータは毎回完璧なショットを打たず、ランダムで照準をずらすように設定している。

8.7 ボールの 3D 表示

ボールの表示は、ball.c の `drawBall` 関数で行っている。`glTranslated` 関数と `glRotated` 関数で座標系を設定した後、`gluSphere` 関数で球を描画している。このとき、`glBindTexture` 関数により、テクスチャをマッピングしている。

8.8 make について

`make` とは、ビルド作業を自動化するプログラムである。Makefile 内にファイルの依存関係などビルドのルールを記述しておくことで、`make` コマンドを実行するだけでビルドすることができる。今回は、ソースファイルの変更に加え、アイコンの画像ファイルとリソースファイルが書き換えられた際も再ビルドが行われるように依存関係を記述した。

9 感想

OpenGL のライブラリである GLUT を用いて、ビリヤードのプログラムを作成した。図形を描画する処理や、ベクトルの計算を行う処理を汎用的な関数にまとめることで、完結にコードを記述できた。

今回のプログラムで一番苦戦したのは、物理演算の処理だった。台と球の摩擦、球同士の衝突、球の回転など、考えることが山ほどあり、恐らくプログラムよりも物理で悩んでいる時間のほうが多かった。1 から物理演算を実装するのは大変だったが、摩擦係数や衝突時の速度のロスなどのパラメータを細かく設定することで、リアルな挙動が実装できた。

また、球の 3D 描画にも挑戦した。2D から 3D になるだけで、視点や光源、陰影など考慮すべき要素が増えた。授業では扱っていない内容だったため、手探りで実装したが、球を 3D で描画することで格段にリッチな表現を実現することができた。

10 謝辞

物理演算を実装するにあたって、物理教員の柳沼先生にアドバイスや文献探しのお手伝いをしていただきました。私の物理の知識だけではこのプログラムを完成させることはできなかったと思います。柳沼先生には大変感謝しております。

参考文献

- [1] 伊藤祥一, “Springs of C 楽しく身につくプログラミング”, 森北出版株式会社, 2017, pp. 109–110.
- [2] “ボール同士の衝突の公式”, 物理学の見つけ方, https://physics-htfi.github.io/classical_mechanics/005.html, 参照 2020/2/3.