

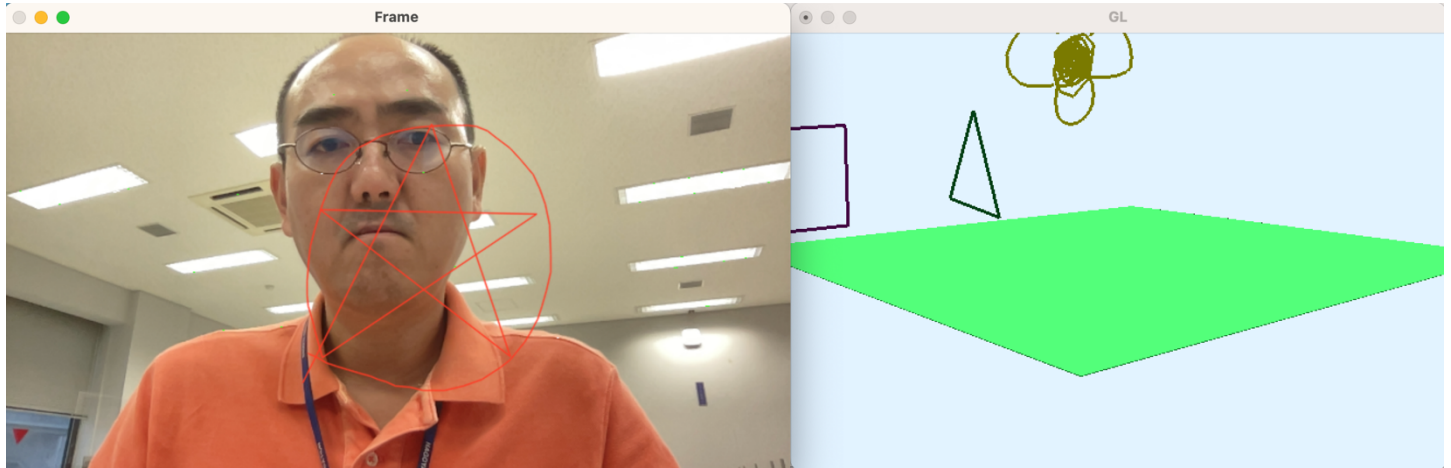
デジタル映像処理及び演習

第14回演習・・・現実空間と仮想空間のインタラクション

インタラクション

ゲームやCGの仮想空間に対するインタラクション（対話操作）の入力デバイスとして、マウスやキーボード、専用のコントローラが一般的である。他にも、直感的なインタフェースとしてジェスチャーがある。骨格を認識する Kinect 等がある。顔の認識は第12回で学んだが、手指などの骨格認識は OpenCV ではサポートされていないので、今回はオプティカルフローを利用したジェスチャー認識及びマウス操作によるインタラクションをやってみよう。

今回やりたいことは、マウス操作で絵を描いて、その絵を CG 空間に転送するインタラクションである。



左側のウィンドウ上にマウスで絵を描き、右側の CG 空間に転送

OpenCV で生成したウィンドウ内のマウスイベントの取得

OpenCV では、生成したウィンドウごとにマウスイベントを取得することができる。マウスイベントの登録は、以下の `setMouseCallback()` 関数を用いる。

```
void cv::setMouseCallback(const string& winname, MouseCallback onMouse, void* userdata=0 )
winname : イベント検知をしたいウィンドウの名前
onMouse : コールバック関数
userdata : onMouse 関数内で使いたいデータ。画像やユーザ定義の構造体など（今回は使用しない）
```

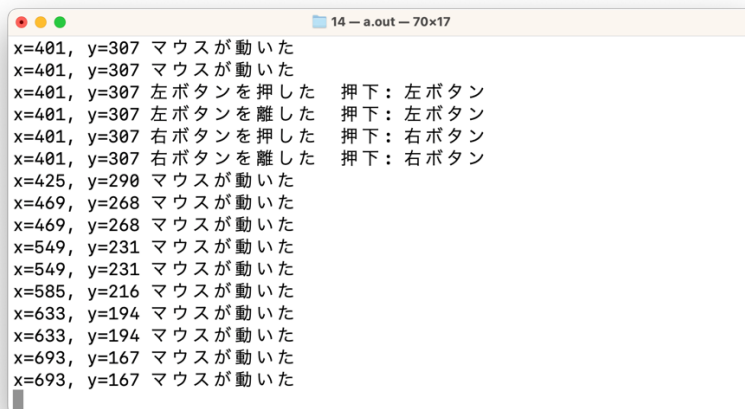
コールバック関数の名前は任意に定義すれば良いが、引数は OpenCV で固定されており以下の通りである。

```
void onMouse(int event, int x, int y, int flags, void* param)
event : 検知されるイベント
    cv::EVENT_MOUSEMOVE          マウスが移動した
    cv::EVENT_LBUTTONDOWN        左ボタンが押された
    cv::EVENT_RBUTTONDOWN        右ボタンが押された
    cv::EVENT_MBUTTONDOWN        中ボタンが押された
    cv::EVENT_LBUTTONUP          左ボタンが離された
    cv::EVENT_RBUTTONUP          右ボタンが離された
    cv::EVENT_MBUTTONUP          中ボタンが離された
    cv::EVENT_LBUTTON_DBLCLK     左ボタンがダブルクリックされた
    cv::EVENT_RBUTTON_DBLCLK     右ボタンがダブルクリックされた
    cv::EVENT_MBUTTON_DBLCLK     中ボタンがダブルクリックされた
x, y : マウスポインタの座標 (x, y) (画像左上隅が原点)
flags : 修飾キーなどの状態
    cv::EVENT_FLAG_LBUTTON       左ボタンが押されている
    cv::EVENT_FLAG_RBUTTON       右ボタンが押されている
    cv::EVENT_FLAG_MBUTTON       中ボタンが押されている
    cv::EVENT_FLAG_CTRLKEY       Ctrl キーが押されている
    cv::EVENT_FLAG_SHIFTKEY      Shift キーが押されている
    cv::EVENT_FLAG_ALTKEY        Alt キーが押されている
param : setMouseCallback の第3引数 userdata と同じ
```

サンプルプログラム"DIP14"とその説明

動作概要

このプログラムは、OpenCV で作成したウィンドウ内のマウスイベントを取ってきて、マウス関係のどのイベントが発生したかを標準出力に表示するものである。initCV() 関数で OpenCV 関係の初期設定の際に setMouseCallback でマウスイベントの登録をしている。マウスイベントが発生した時に呼ばれるコールバック関数 mouseCallback で、マウスの位置やマウスボタンの状態等を取得している。



```
x=401, y=307 マウスが動いた
x=401, y=307 マウスが動いた
x=401, y=307 左ボタンを押した 押下: 左ボタン
x=401, y=307 左ボタンを離した 押下: 左ボタン
x=401, y=307 右ボタンを押した 押下: 右ボタン
x=401, y=307 右ボタンを離した 押下: 右ボタン
x=425, y=290 マウスが動いた
x=469, y=268 マウスが動いた
x=469, y=268 マウスが動いた
x=549, y=231 マウスが動いた
x=549, y=231 マウスが動いた
x=585, y=216 マウスが動いた
x=633, y=194 マウスが動いた
x=633, y=194 マウスが動いた
x=693, y=167 マウスが動いた
x=693, y=167 マウスが動いた
```

ソースコード (DIP14)

```
#include <iostream> //入出力関連ヘッダ
#include <GLUT/glut.h> //OpenGL
#include <math.h> //数学関数
#include <opencv2/opencv.hpp> //OpenCV関連ヘッダ

//関数名の宣言
.....

void initCV(void);
void mouseCallback(int event, int x, int y, int flags, void *userdata);

//グローバル変数
.....

cv::Mat originalImage, frameImage; //画像格納用
double theta = 0.0;
double delta = 1.0; // 回転の速度
int rotFlag = 1; // 回転フラグ

//main関数
int main(int argc, char* argv[])
{
    //OpenGL初期化
    glutInit(&argc, argv);

    //OpenCV初期設定処理
    initCV();

    //OpenGL初期設定処理
    initGL();

    //イベント待ち無限ループ
    glutMainLoop();

    return 0;
}

//OpenCV初期設定処理
void initCV(void)
{
    //①ビデオキャプチャの初期化
```

```

capture = cv::VideoCapture(0); //カメラ0番をオープン
if (capture.isOpened()==0) { //オープンに失敗した場合
    printf("Capture not found\n");
    return;
}

//②画像格納用インスタンス準備
int imageWidth=720, imageHeight=405;
imageSize = cv::Size(imageWidth, imageHeight); //画像サイズ
frameImage = cv::Mat(imageSize, CV_8UC3); //3チャンネル

//③画像表示用ウィンドウの生成
cv::namedWindow("Frame");
cv::moveWindow("Frame", 0, 0);

//マウスコールバック関数のウィンドウへの登録
cv::setMouseCallback("Frame", mouseCallback);
}

.....
//ディスプレイコールバック関数
void display()
{
    // ----- OpenCV -----
    // (a) ビデオキャプチャから1フレーム"originalImage"を取り込んで, "frameImage"を生成
    capture >> originalImage;
    //ビデオが終了したら無限ループから脱出
    if (originalImage.data==NULL) {
        exit(0);
    }
    //"originalImage"をリサイズして"frameImage"生成
    cv::resize(originalImage, frameImage, imageSize);

    // 描いた図形を描画 (cv::line等で)

    // (b) "frameImage"の表示
    cv::imshow("Frame", frameImage);

    // ----- OpenGL -----
    GLfloat col[4]; //色設定用

    //ウィンドウ内消去
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    //行列初期化
    glLoadIdentity();

    //視点座標の計算
    double ex = eDist*cos(eDegX*M_PI/180.0)*sin(eDegY*M_PI/180.0);
    double ey = eDist*sin(eDegX*M_PI/180.0);
    double ez = eDist*cos(eDegX*M_PI/180.0)*cos(eDegY*M_PI/180.0);

    //視点視線の設定
    gluLookAt(ex, ey, ez, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0); //変換行列に視野変換行列を乗算

    //光源0の位置指定
    GLfloat pos0[] = {200.0, 700.0, 200.0, 0.0}; //(x, y, z, 0(平行光源)/1(点光源))
    glLightfv(GL_LIGHT0, GL_POSITION, pos0);

    //----- 地面 -----
    //色設定
    col[0] = 0.5; col[1] = 1.0; col[2] = 0.5; // (0.5, 1.0, 0.5) : RGB
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, col); //拡散反射係数
    col[0] = 1.0; col[1] = 1.0; col[2] = 1.0; col[3] = 1.0;
    glMaterialfv(GL_FRONT, GL_SPECULAR, col);
    glMaterialf(GL_FRONT, GL_SHININESS, 64); //ハイライト係数
    glPushMatrix(); //行列一時保存
    glScaled(1000, 1, 1000); //拡大縮小

```

```

glutSolidCube(1.0); //立方体の配置
glPopMatrix(); //行列復帰

// 図形描画(OpenCVで描画した絵(取得したマウス座標(cv::Point型の軌跡)をcv::Point3f型にする)

//描画実行
glutSwapBuffers();
}

.....

//キーボードコールバック関数(key:キーの種類, x,y:座標)
void keyboard(unsigned char key, int x, int y)
{
    switch (key) {
        case 'q':
        case 'Q':
        case 27:
            exit(0);
    }
}

// マウスコールバック関数 in a window made by OpenCV
void mouseCallback(int event, int x, int y, int flags, void *userdata)
{
    // マウスの座標を出力
    std::cout << "x=" << x << ", y=" << y << " ";

    // イベントの種類を出力
    switch (event) {
        case cv::EVENT_MOUSEMOVE:
            std::cout << "マウスが動いた";
            break;
        case cv::EVENT_LBUTTONDOWN:
            std::cout << "左ボタンを押した";
            break;
        case cv::EVENT_RBUTTONDOWN:
            std::cout << "右ボタンを押した";
            break;
        case cv::EVENT_LBUTTONUP:
            std::cout << "左ボタンを離した";
            break;
        case cv::EVENT_RBUTTONUP:
            std::cout << "右ボタンを離した";
            break;
        case cv::EVENT_RBUTTONDOWNCLK:
            std::cout << "右ボタンをダブルクリック";
            break;
        case cv::EVENT_LBUTTONDOWNCLK:
            std::cout << "左ボタンをダブルクリック";
            break;
    }

    // マウスボタンと特殊キーの押下状態を出力
    std::string str;
    if (flags & cv::EVENT_FLAG_ALTKEY) {
        str += "Alt "; // ALTキーが押されている
    }
    if (flags & cv::EVENT_FLAG_CTRLKEY) {
        str += "Ctrl "; // Ctrlキーが押されている
    }
    if (flags & cv::EVENT_FLAG_SHIFTKEY) {
        str += "Shift "; // Shiftキーが押されている
    }
    if (flags & cv::EVENT_FLAG_LBUTTON) {
        str += "左ボタン "; // マウスの左ボタンが押されている
    }
    if (flags & cv::EVENT_FLAG_RBUTTON) {
        str += "右ボタン"; // マウスの右ボタンが押されている
    }
}

```

```
if (!str.empty()) {  
    std::cout << "  押下: " << str;  
}  
std::cout << std::endl;  
  
}
```

課題1 マウスで図形を描画

マウスイベントで取得したマウス座標を直線 (`cv::line`) で繋いで, “Frame”ウィンドウ上に図形を描画せよ. 描画の様子がわかる10秒程度の動画を提出すること.

(ヒント)

1. マウス座標を覚えておく配列を `cv::Point` 型で用意, グローバル変数にすること. 現在何点入力されているかを覚えておく変数も用意すること. できる人は `std::vector` で動的配列にしても良い.
2. クリックした座標を配列に追加
3. `cv::line()` を利用してウィンドウ上に線を描画. `cv::line()` については第7回講義資料を参照.

課題2 ジェスチャー認識

オプティカルフローを利用して, 手で左右にスワイプするジェスチャーを認識せよ. 右スワイプなら「右にスワイプしました」, 左スワイプなら「左にスワイプしました」と標準出力に出力しなさい. スワイプ動作と出力が分かる10秒程度の動画を提出すること.

課題3 マウスで描いた図形を CG 空間へ転送

マウスで描いた図形をスワイプジェスチャーで CG 空間に転送, あるいは破棄できるようにしなさい. 例えば, 右スワイプの時に転送 (CG 空間に描いた図形が出現), 左スワイプの時に描いた図形の破棄 (やり直し). 動作が分かる10秒程度の動画を提出すること.

(ヒント)

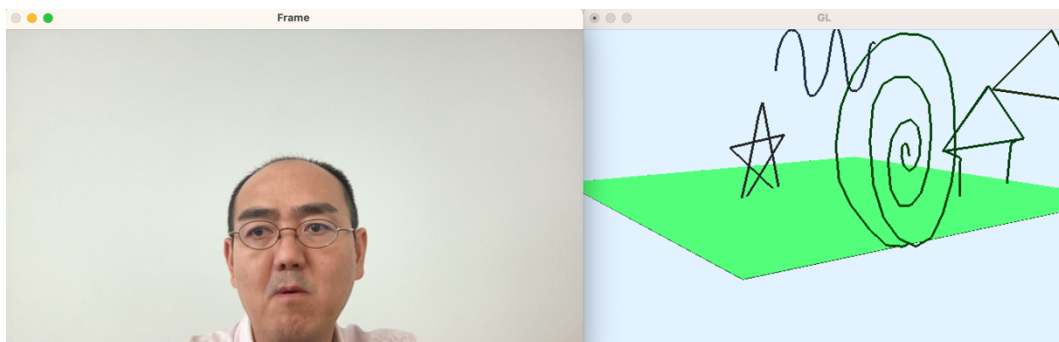
1. 描いた図形は `cv::Point` 型の配列で覚えている. `cv::Point` 型は (x, y) の2次元座標なので, (x, y, z) の3次元座標に変換すること. `cv::Point3f` 型の配列を用いると良い. CG空間のどこに配置するかは各自で決めれば良いが, ランダムな位置に置いてよい. 乱数の使い方は付録1を参照.
2. CG空間に転送した図形の色は決まった色で良いが, ランダムに変わっても良い.
3. マウスの座標系は画像上から下方向がY軸正方向であるが, CG空間では画面下から上方向が正方向であることに注意

課題4 複数の図形を転送

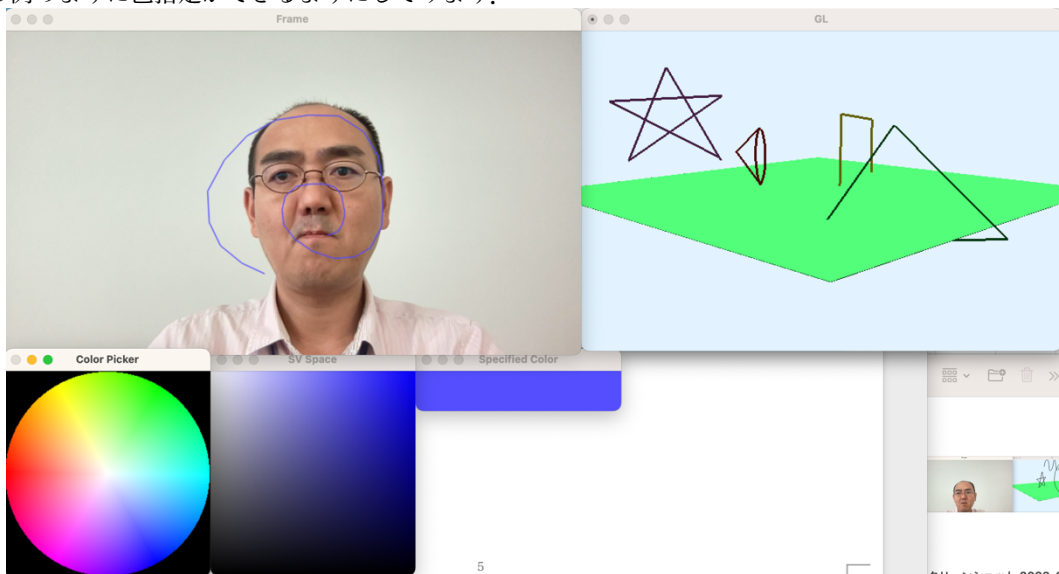
図形の転送を複数回実行できるようにプログラムを修正しなさい. 複数の絵が転送されていることが分かる15~20秒程度の動画を提出しなさい. ドラッグで図形の頂点を追加すれば, フリーハンドで絵が描ける. フリーハンドの図形も最低1つは含むこと. 転送した各図形に何かしらの動きが付与されていることが望ましい.

(ヒント)

1. マウスで描画した図形 (`cv::Point` 型の配列) と CG 空間に転送した図形 (`cv::Point3f` 型の配列) が1対1の対応では, 複数回転送できない. CG空間の図形を1次元配列から2次元配列にする必要がある. つまり, 過去に転送した図形を覚えておく必要がある.



できる人は, 下の例のように色指定ができるようにしてみよう.



付録1 乱数

乱数は `stdlib.h` にある `rand()` 関数で発生させることができる。C++では、より良質な乱数を生成でき、扱い方も比較的簡単になっているので紹介する。

```
=====
#include <iostream> //入出力関連ヘッダ
#include <GLUT/glut.h> //OpenGL
#include <math.h> //数学関数
#include <opencv2/opencv.hpp> //OpenCV関連ヘッダ
#include <random>

. . . . .

//グローバル変数
. . . . .

std::random_device rnd;
std::mt19937 mt(rnd());
std::uniform_int_distribution<> RandX(-500, 500);
std::uniform_real_distribution<> RandSP(1.0, 10.0);
=====
```

`random` をインクルードして、`std::random_device` で乱数用変数を用意する。次に、

```
std::mt19937 mt(rnd());
```

で乱数発生アルゴリズムを選び、変数 `mt` として宣言する。そして例えば、`x` 座標を `[-500, 500]` の間でランダムに整数を生成する場合は、

```
std::uniform_int_distribution<> RandX(-500, 500);
```

と記述する。実際に生成するときは、

```
int tx = RandX(mt);
```

のように書けば良い。

実数の生成は、`uniform_int_distribution` の代わりに、

```
std::uniform_real_distribution<> RandSP(1.0, 10.0);
```

を使えば、上の例だと `[1.0, 10.0]` の範囲で実数を生成する。使い方は `RandX` の時と同様に、

```
double speed = RandSP(mt);
```

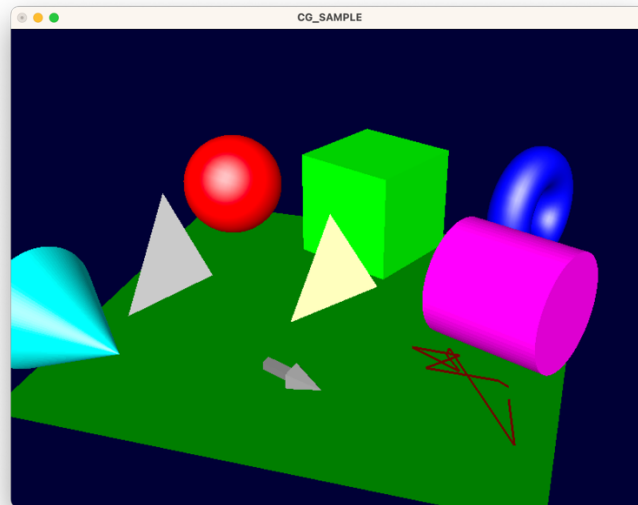
と書けば良い。

付録2 OpenGL における図形の描画

2KX の「CG プログラミング及び演習」を受けていない人は、OpenGL での図形の描画方法でつまづくかもしれない。以下に今回使う描画方法を“cg_sample.cpp”を基に簡単に解説する。

描画

`display()` 関数に記述されているオブジェクトが描画される。サンプルでは球など9種類描画している。



与えられた頂点を結ぶ図形 (`glBegin()` ~ `glEnd()`)

与えられた頂点を結ぶ図形は、`glBegin(PARAMETER) ~ glEnd()` で描画される (L.587 - L.593)。ここで、`PARAMETER` はどのように頂点を結ぶかを指定するパラメータであり、ここでは `GL_LINE_STRIP` を用いて折れ線で描画している。そのほか、`GL_LINE_LOOP` や `GL_TRIANGLES` など様々用意されている。

頂点を覚えておく配列を用意し (L.40, L41), `initGL` 関数で各要素にランダムに座標を入れている。できる人は、`std::vector` で動的配列を使っても良い。