



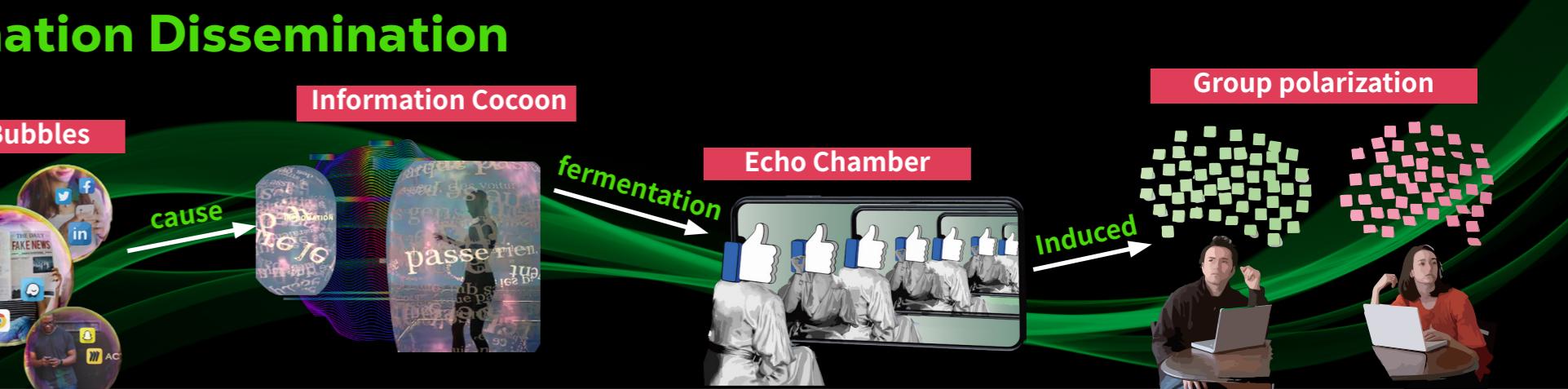
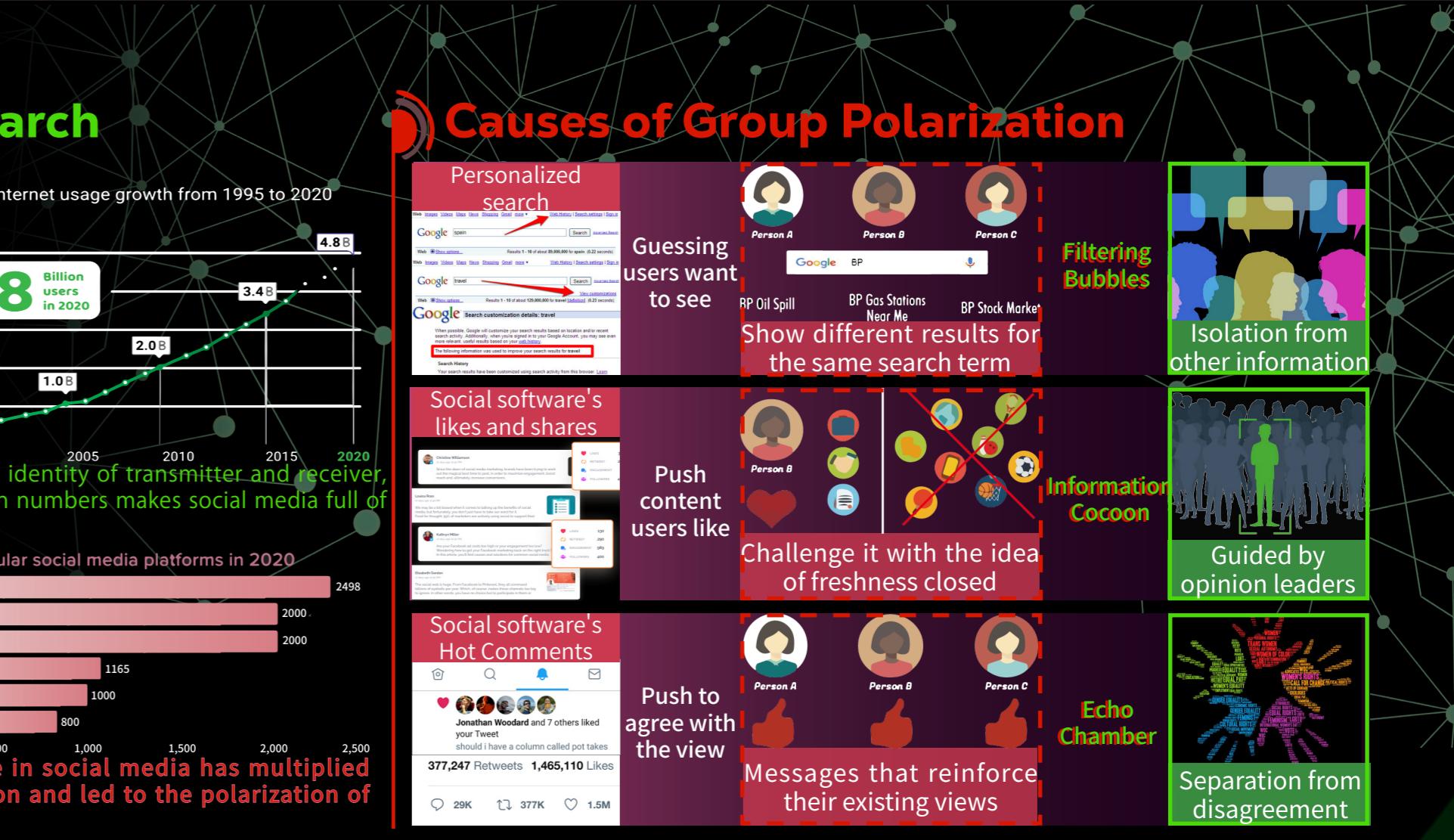
Information Social

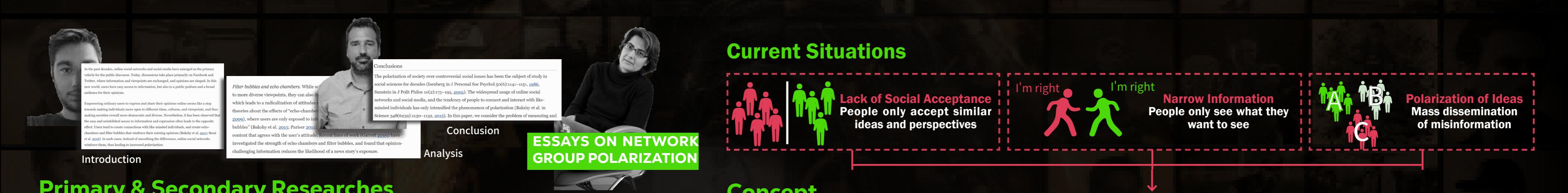
In the era of social media, the access to information increases, and the filtering in the process of information transmission will promote the formation of different circles, and different circles will form mutual closure or even opposition due to different factors such as information acceptance, interests and personality traits.

Case Studies

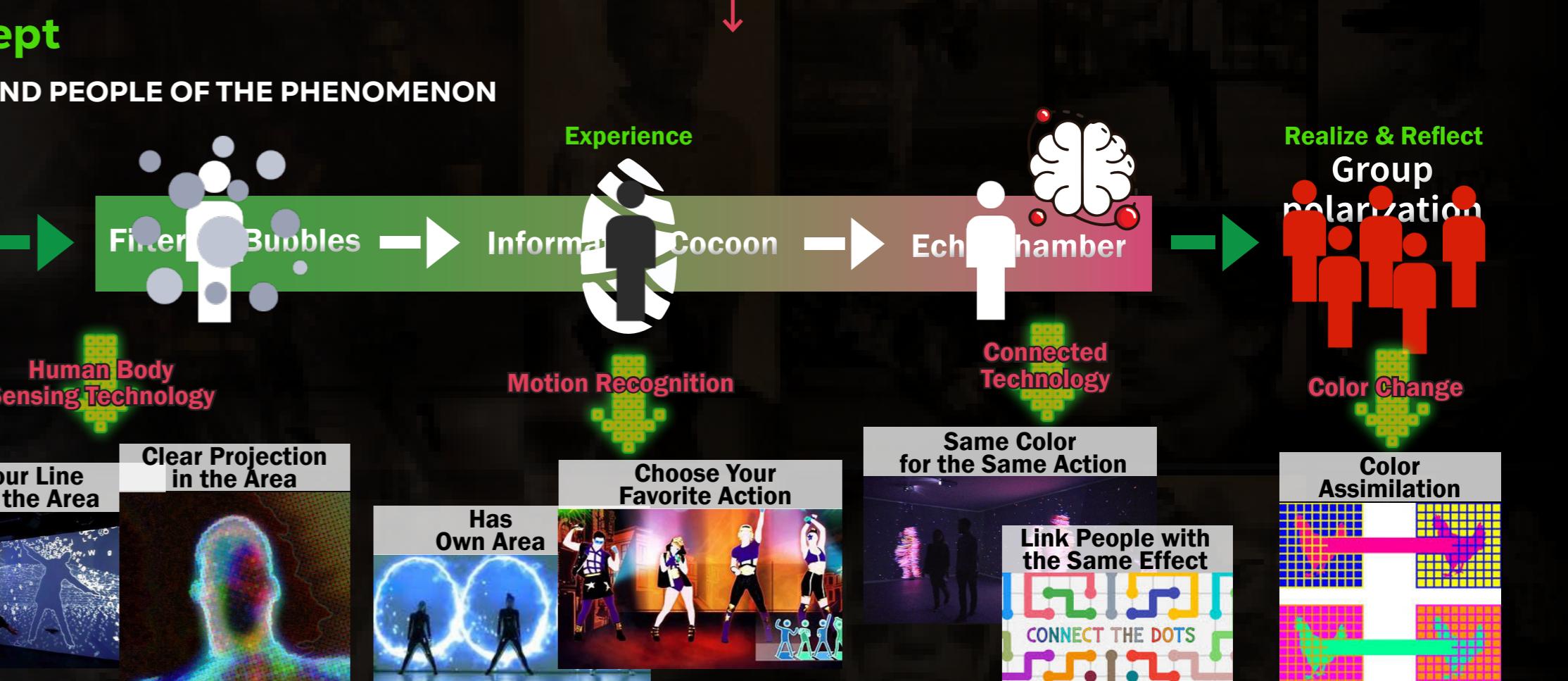
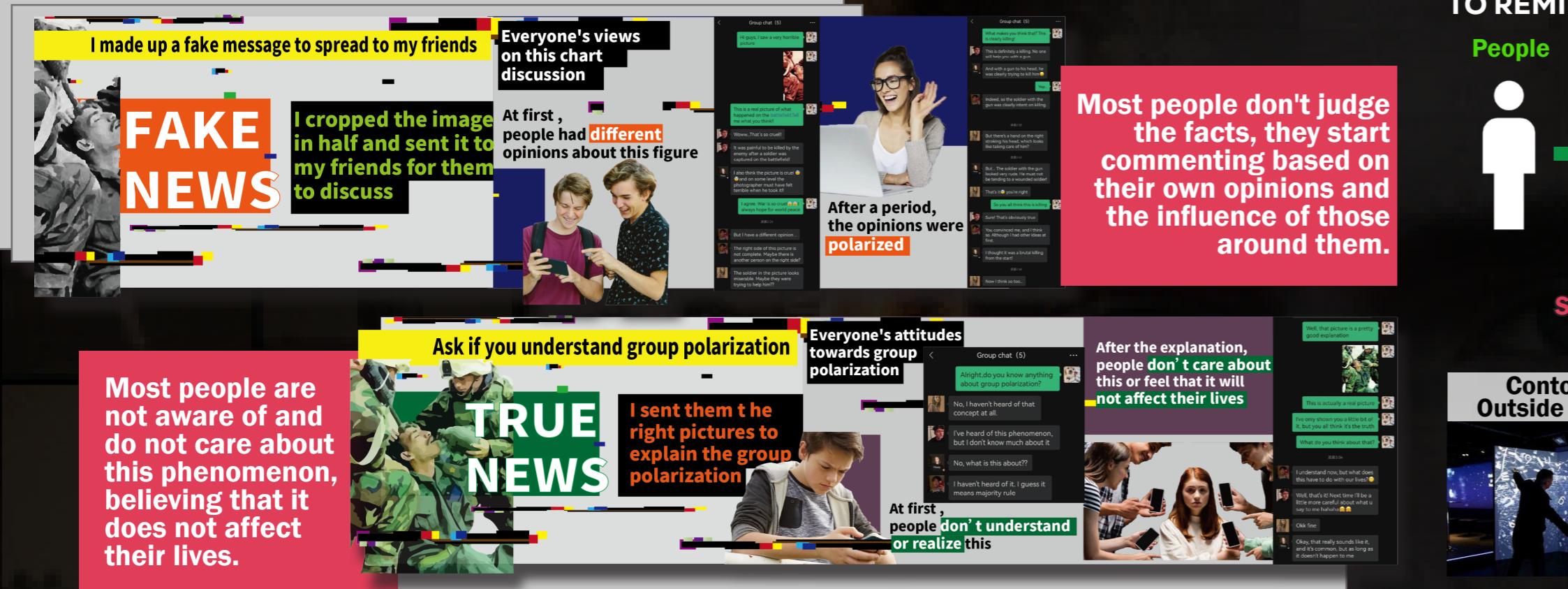
REAL NEWS

Internet Violence



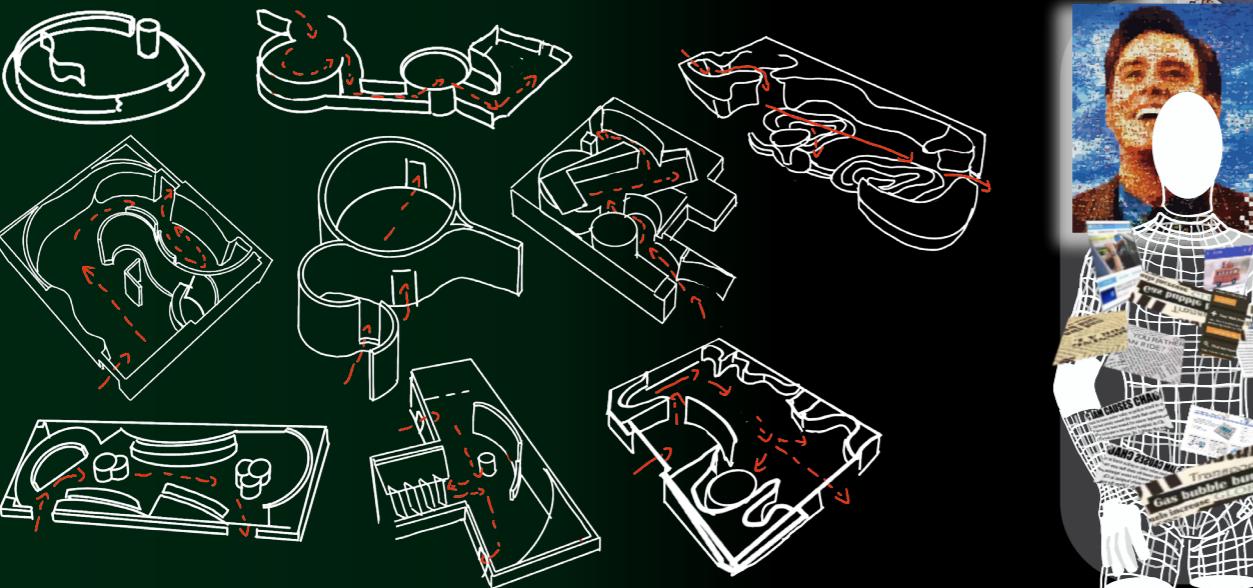


Primary & Secondary Researches



Sketches & Modeling

EXHIBITION HALL DESIGN



Idea 1

Space utilization ●●

Display area ●●●

Movement line ●●

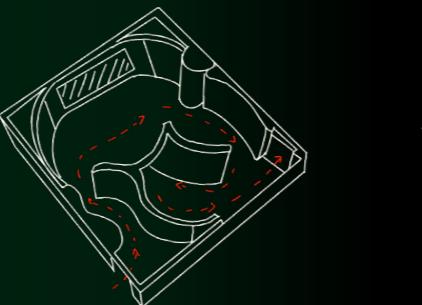


Idea 2

Space utilization ●●●

Display area ●●●

Movement line ●●

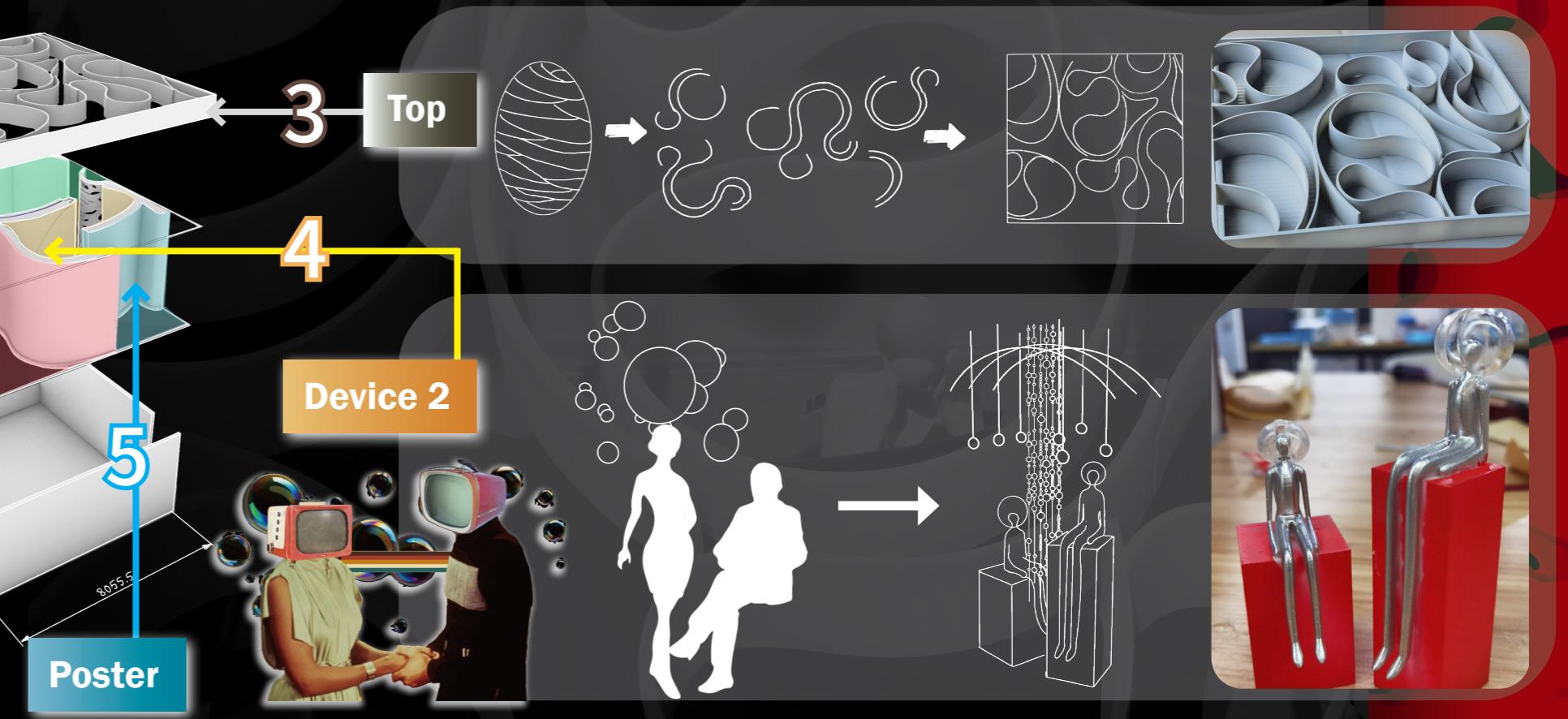
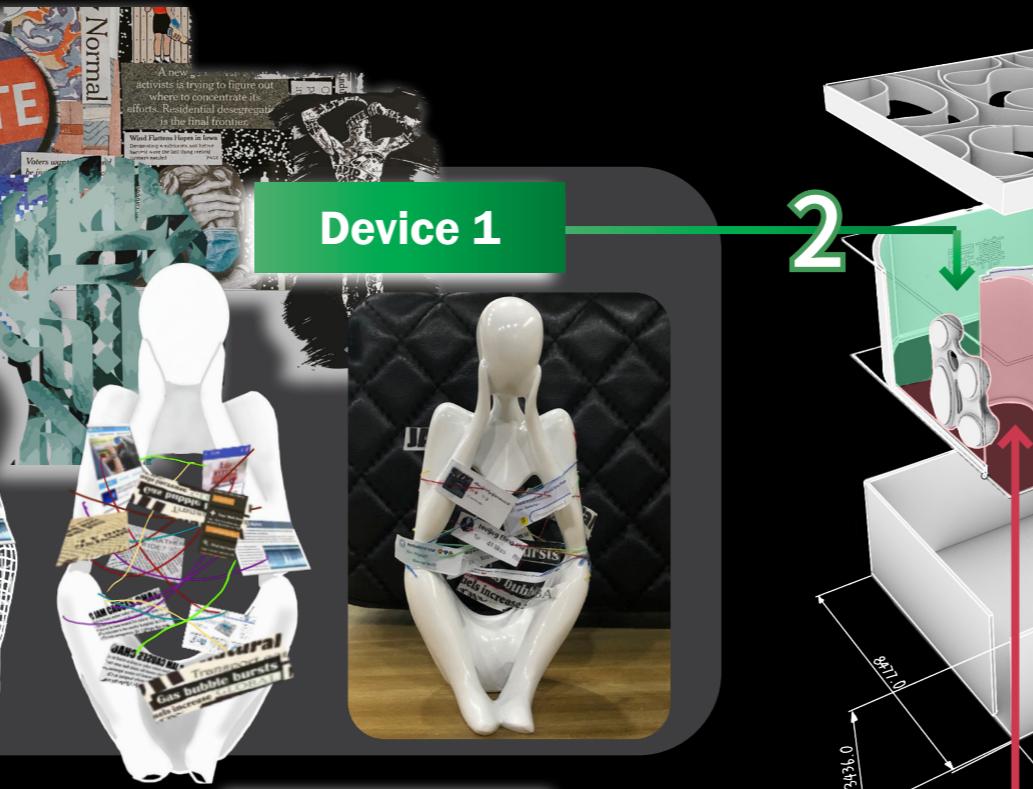
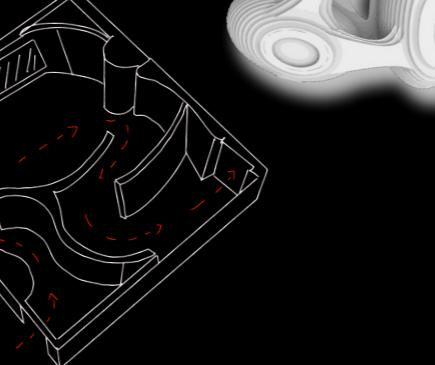


Final Design

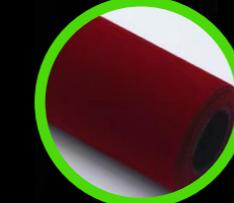
Space utilization ●●●●

Display area ●●●●

Movement line ●●●●



MATERIALS



3D printing
resin

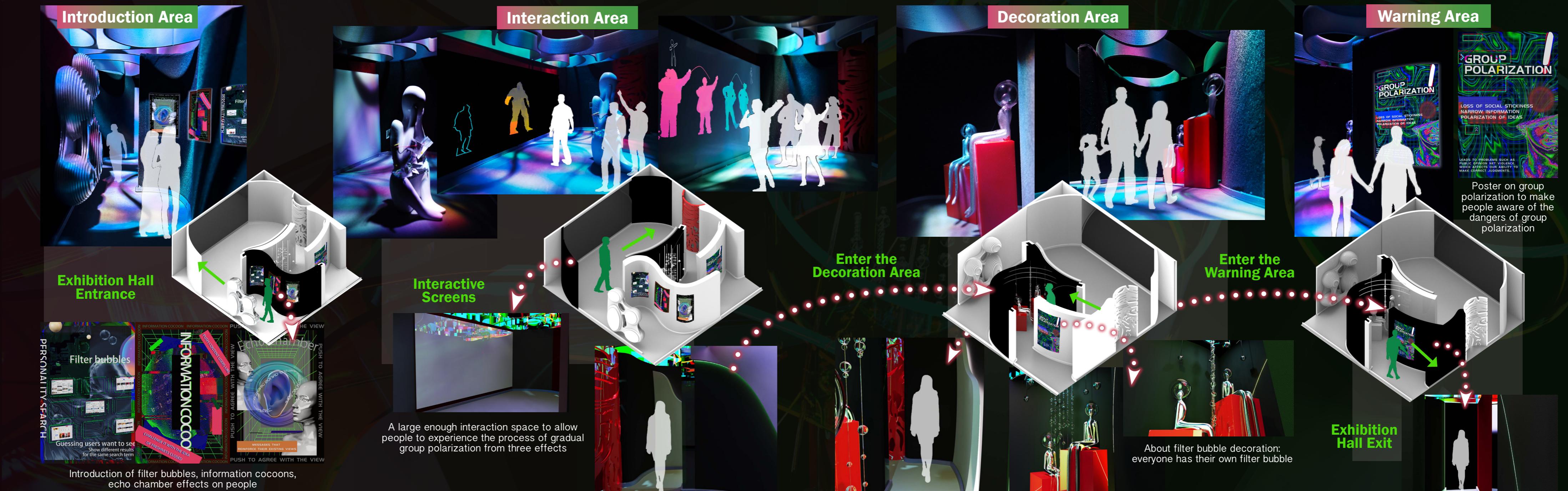


PVC
sheet

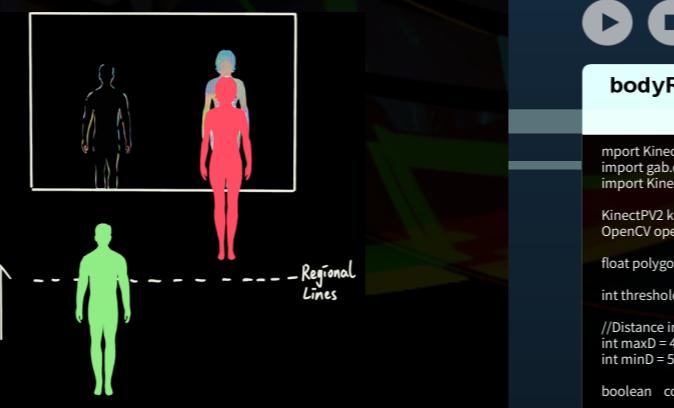


Flannel
stickers





Scenes in Interaction Area



Motion Recognition Programming

```

bodyRelation1 drawUtils

import KinectPV2.KJoint;
import gab.opencv.*;
import KinectPV2.*;

KinectPV2 kinect;
OpenCV opencv;

float polygonFactor = 3;
int threshold = 200;

//Distance in cm
int maxD = 4500; //4.5m
int minD = 50; //50cm

boolean contourBodyIndex = false;

int Y_AXIS = 1;
int X_AXIS = 2;
color b1, b2, c1, c2;

float zoff = 0;

PGraphics pg;

PImage gestureImage=new PImage[4];
PVector[] pt=new PVector[2];
int index=0;

float[] actThresh={1.0, 9.5, 8.6, 0.44};
float[] actThresh1={0.7, 0.6, 0.5, 0.44};
float[] action1={2.3412, -0.594, -0.594, -1.244, -1.957, 1.557, 1.584};
float[] action2={-2.832, -0.213, -1.007, -2.0803, 1.729, 1.314, 1.729, 1.314};
float[] action3={1.927, -0.503, -2.703, -1.751, 1.853, 0.983, 1.853, 1.532};
float[] action4={2.339, 1.1801, 1.168, 2.099, 1.6001, 0.992, 1.1692, 0.992};

float[] angles=new float[8];
ArrayList<PVector> redLine=new ArrayList<PVector>;
ArrayList<PVector> greenLine=new ArrayList<PVector>;

void setup(){
    size(1920, 1080, P2D);
    opencv = new OpenCV(this, 512, 424);
    //fullScreen(P2D);
    pt[0]=new PVector(10, 0);
    pt[1]=new PVector(10, 10);
    pg = createGraphics(512, 424);
    cols = floor(width / scl);
    rows = floor(height / scl);
    println(cols * rows);
    kinect = new KinectPV2(this);
    kinect.enableSkeletonColorMap(true);
    kinect.enableColor(true);
    kinect.enableDepth(true);
    kinect.enableBodyTracking(true);
    kinect.enableDepthMasking(true);
    kinect.enableSkeletonDepthMap(true);
    kinect.init();
    b1 = color(255);
    b2 = color(0);
    c1 = color(240, 57, 87);
    c2 = color(69, 117, 161);
    background(255);
}

background(255);
colorMode(RGB);
for (int i=0; i<4; i++) {
    gestureImage[i]=loadImage(str[i]+".png");
}

KinectPV2 kinect;
OpenCV opencv;

float polygonFactor = 3;
int threshold = 200;

//Distance in cm
int maxD = 4500; //4.5m
int minD = 50; //50cm

boolean contourBodyIndex = false;

int Y_AXIS = 1;
int X_AXIS = 2;
color b1, b2, c1, c2;

float zoff = 0;

PGraphics pg;

PImage gestureImage=new PImage[4];
PVector[] pt=new PVector[2];
int index=0;

float[] actThresh={1.0, 9.5, 8.6, 0.44};
float[] actThresh1={0.7, 0.6, 0.5, 0.44};
float[] action1={2.3412, -0.594, -0.594, -1.244, -1.957, 1.557, 1.584};
float[] action2={-2.832, -0.213, -1.007, -2.0803, 1.729, 1.314, 1.729, 1.314};
float[] action3={1.927, -0.503, -2.703, -1.751, 1.853, 0.983, 1.853, 1.532};
float[] action4={2.339, 1.1801, 1.168, 2.099, 1.6001, 0.992, 1.1692, 0.992};

float[] angles=new float[8];
ArrayList<PVector> redLine=new ArrayList<PVector>;
ArrayList<PVector> greenLine=new ArrayList<PVector>;

void setup(){
    size(1920, 1080, P2D);
    opencv = new OpenCV(this, 512, 424);
    //fullScreen(P2D);
    pt[0]=new PVector(10, 0);
    pt[1]=new PVector(10, 10);
    pg = createGraphics(512, 424);
    cols = floor(width / scl);
    rows = floor(height / scl);
    println(cols * rows);
    kinect = new KinectPV2(this);
    kinect.enableSkeletonColorMap(true);
    kinect.enableColor(true);
    kinect.enableDepth(true);
    kinect.enableBodyTracking(true);
    kinect.enableDepthMasking(true);
    kinect.enableSkeletonDepthMap(true);
    kinect.init();
    b1 = color(255);
    b2 = color(0);
    c1 = color(240, 57, 87);
    c2 = color(69, 117, 161);
    background(255);
}

noFill();
strokeWeight(4);
stroke(255, 0, 0);
fill(255, 0, 0);
//line(pt[0].x, pt[0].y, pt[1].x, pt[1].y);
///point(pt[0].x, pt[0].y);
///point(pt[1].x, pt[1].y);
//text1(pt[0].x, pt[0].y);
//text2(pt[1].x, pt[1].y);

c = lerpColor(#2af598, #151f6d, inter);
} else {
    c = lerpColor(#3EECAC, #EE74E1, inter);
}
stroke(c);
strokeWeight(8);
line(sPt.x*width/512.0, sPt.y*height/424.0, tWidth/512.0, tHeight/424.0);
flag=0;
}

//noTint();
ArrayList<KSkeleton> skeletonArray = kinect.getSkeletonColorMap();
markColor[0]=-1;
markColor[1]=-1;
markColor[2]=-1;
markColor[3]=-1;
/individual JOINTS
for (int i=0; i<skeletonArray.size(); i++) {
    KSkeleton skeleton = skeletonArray.get(i);
    if (skeleton.isTracked()) {
        KJoint[] joints = skeleton.getJoints();
        color col = skeleton.getIndexColor();
        fill(col);
        stroke(col);
        drawBody(i, joints);
    }
    curveVertex(greenLine.get(i).x, greenLine.get(i).y);
    curveVertex(greenLine.get(greenLine.size()-1).x, greenLine.get(i).y);
    get(greenLine.size()-1);
    curveVertex(greenLine.get(greenLine.size()-1).x, greenLine.get(greenLine.size()-1).y);
    endShape();
}
stroke(#45b1a1);
if (greenLine.size()>1) {
beginShape();
curveVertex(redLine.get(0).x, redLine.get(0).y);
for (int i=0; i<redLine.size()-1; i++) {
    curveVertex(redLine.get(redLine.size()-1).x, redLine.get(redLine.size()-1).y);
    curveVertex(redLine.get(redLine.size()-1).x, redLine.get(redLine.size()-1).y);
    endShape();
}
}

//draw different color for each hand state
drawHandState(joints[KinectPV2.JointType_HandRight]);
drawHandState(joints[KinectPV2.JointType_HandLeft]);
int id=-1;
if (joints[KinectPV2.JointType_Head].getX()<width/4) {
    id=0;
} else if (joints[KinectPV2.JointType_Head].getX()<width*2/4) {
    id=1;
} else if (joints[KinectPV2.JointType_Head].getX()<width*3/4) {
    id=2;
} else {
    id=3;
}
boolean counterF=true;
void keyPressed() {
    if (key=='a') {
        counterF=!counterF;
    }
}
void setGradient(int x, int y, float w, color c1, color c2, int axis) {
    if (axis == Y_AXIS) { // Top to bottom gradient
        for (int i = y; i <= y+w; i++) {
            float inter = map(i, y, y+w, 0, 1);
            color c = lerpColor(c1, c2, inter);
            stroke(c);
            line(x, i, x+w, i);
        }
    } else if (axis == X_AXIS) { // Left to right gradient
        for (int i = x; i <= x+w; i++) {
            float inter = map(i, x, x+w, 0, 1);
            color c = lerpColor(c1, c2, inter);
            stroke(c);
            line(i, y, i, y+w);
        }
    }
}

strokeWeight(4);
stroke(255, 200, 200);
line(width/4, 0, width/2, height);
line(width/2, 0, width/4, height);
line(width/4, 0, width/3/4, height);
line(width/3/4, 0, width/4, height);
noFill();
stroke(#f03957);
}

```