Streck Graph

Florian Schäfers 2077840
Lezgin Turgut 2077569

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

INHALTSVERZEICHNIS

1. EINLEITUNG

Im Rahmen des Studiums Media Systems hatten wir (Florian Schäfers und Lezgin Turgut) die Aufgabe in der Vorlesung “Mobile Systeme”, unter der Leitung von Prof. Dr. Andreas Plaß, eine mobile App mit dem Schwerpunkt Messung zu erstellen. Wir entschieden uns mit unserer App mittels Global Positioning System (GPS) die Wegstrecke sowie die aktuelle Bewegungsgeschwindigkeit zu messen. Zusätzlich soll der zurückgelegte Weg grafisch dargestellt werden. Programmiert wurde sie für das Betriebssystem Android.

2. AUFBAU DER APP

Die Android- App “Streck0Graph” misst die zurückgelegte Strecke und die Durchschnittsgeschwindigkeit. Ebenfalls wird die Himmelsrichtung des Aktuellen Standorts im Bezug zur Ausgangsposition angezeigt. Die Geschwindigkeit, die zurückgelegte Strecke und die Richtung werden auf dem Bildschirm angezeigt. Die zurückgelegte Strecke wird zusätzlich visualisiert dargestellt. Die Standort Daten werden mithilfe der Google Location API abgefragt. Zur Visualisierung wurde OpenGL 2.0 verwendet.

Google Play Service

Der Google Play Services ist eine SDK und API für Android Geräte. Sie bietet dem Entwickler die Möglichkeit zahlreichen Google Funktionen in seine App zu integrieren. Dazu gehören Account Synchronisation, Google+, Google Maps, Location API, Google Play Games, Cloud Messaging, Android Device Manager und noch mehr.
Für unsere App benötigten wir jedoch nur die Location API. Mit ihr war es uns möglich, genauere Standorte zu erhalten.

Open GL

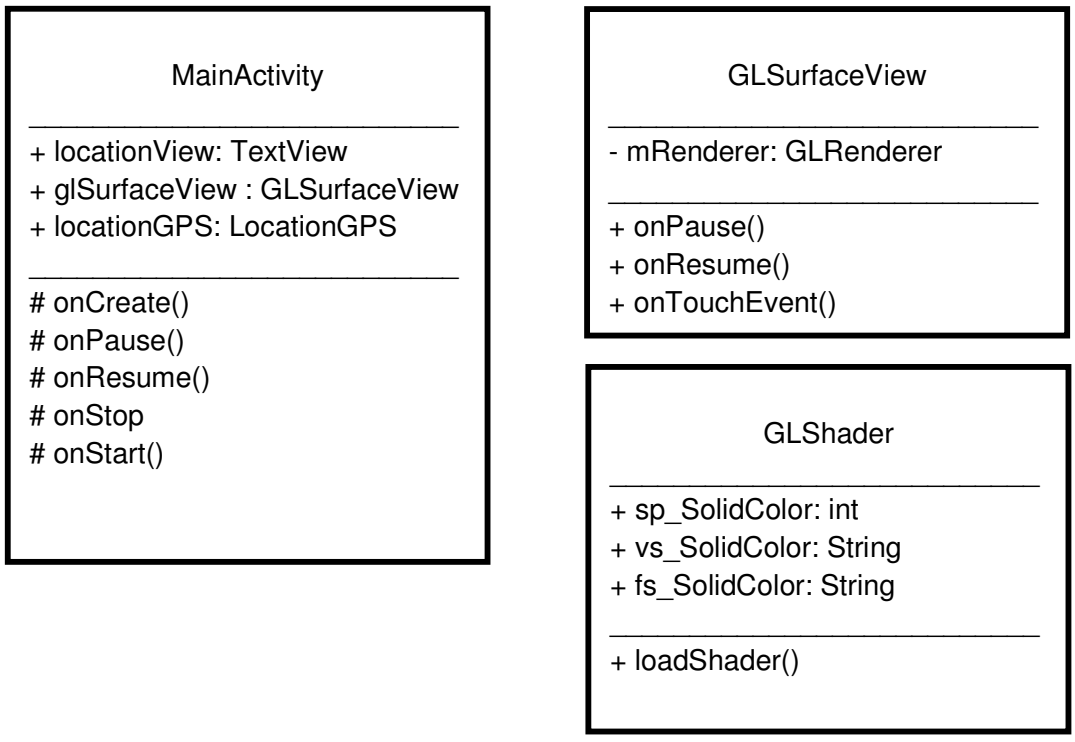
Die Open Graphics Library ist eine Spezifikation für eine Plattform- und programmiersprachenunabhängige Programmierschnittstelle zur Entwicklung von 2D- und 3D-Computergrafikanwendungen.
Android selber unterstützt folgende Open GL ES API:

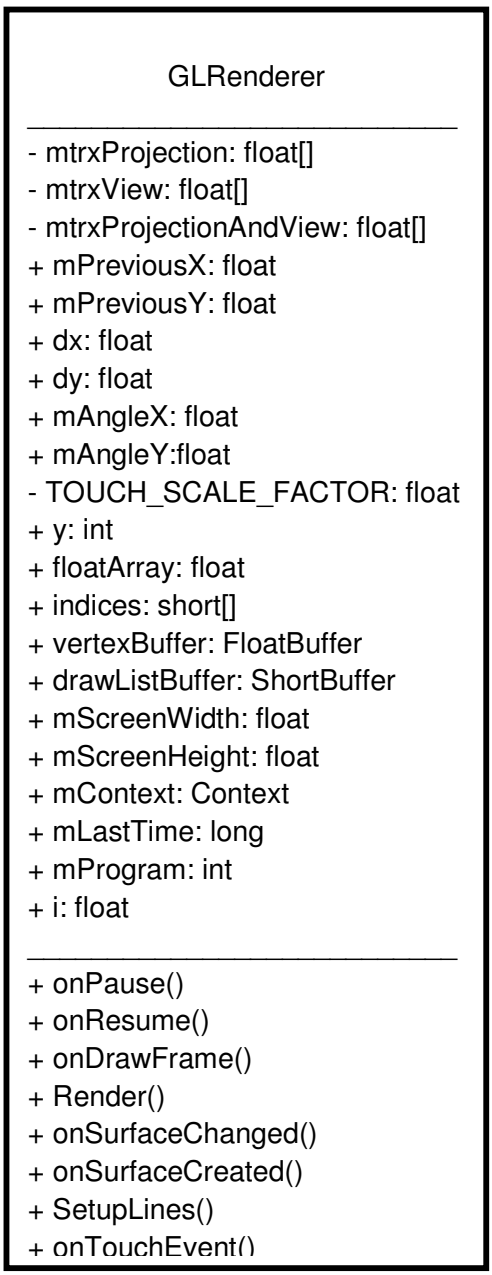
- OpenGL ES 1.0/1.1 - Unterstüzt von Android 1.0 und höher
- OpenGL ES 2.0 - Unterstüzt von Android 2.2 (API Level 8.0) und höher
- OpenGL ES 3.0 - Unterstüzt von Android 4.3 (API Level 18.0) und höher

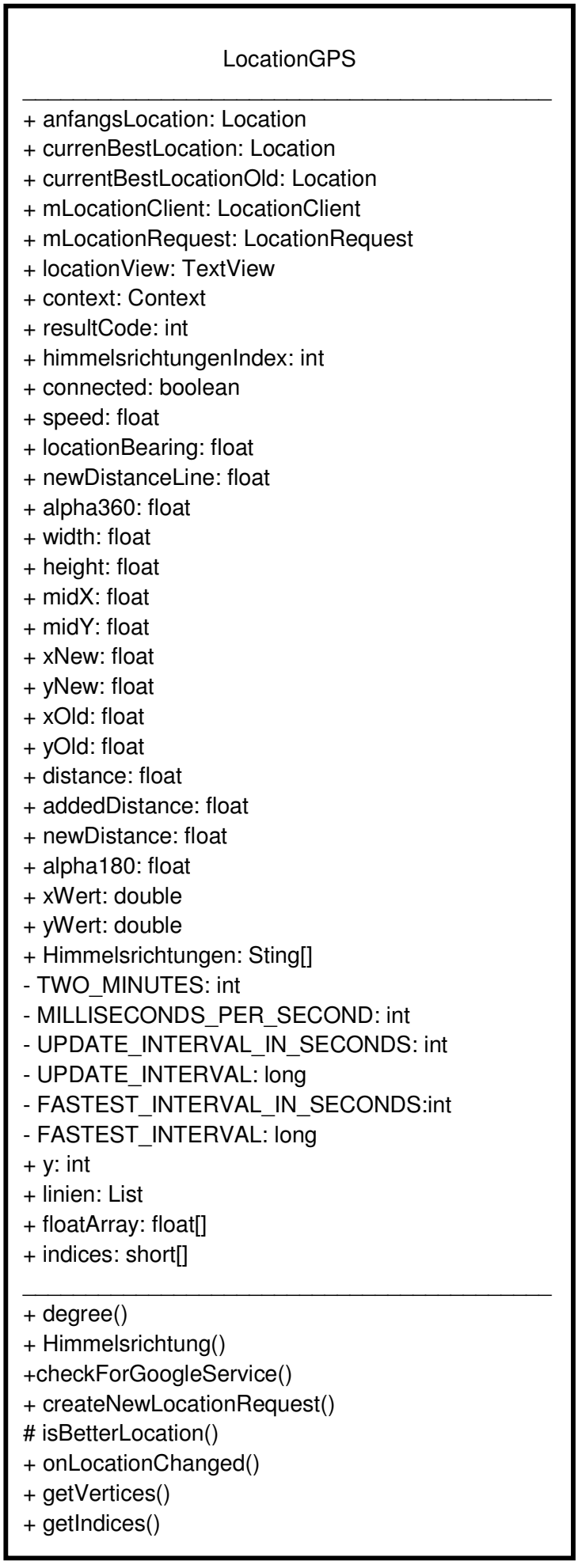
In Unserer Anwendung wurde Open GL 2.0 verwendet, da es zurzeit das gängigste ist.

Quelltext

Klassen Übersicht:




GLRenderer
- mtrxProjection: float[]
- mtrxView: float[]
- mtrxProjectionAndView: float[]
+ mPreviousX: float
+ mPreviousY: float
+ dx: float
+ dy: float
+ mAngleX: float
+ mAngleY:float
- TOUCH_SCALE_FACTOR: float
+ y: int
+ floatArray: float
+ indices: short[]
+ vertexBuffer: FloatBuffer
+ drawListBuffer: ShortBuffer
+ mScreenWidth: float
+ mScreenHeight: float
+ mContext: Context
+ mLastTime: long
+ mProgram: int
+ i: float
+ onPause()
+ onResume()
+ onDrawFrame()
+ Render()
+ onSurfaceChanged()
+ onSurfaceCreated()
+ SetupLines()
+ onTouchEvent()


LocationGPS
+ anfangsLocation: Location
+ currenBestLocation: Location
+ currentBestLocationOld: Location
+ mLocationClient: LocationClient
+ mLocationRequest: LocationRequest
+ locationView: TextView
+ context: Context
+ resultCode: int
+ himmelsrichtungenIndex: int
+ connected: boolean
+ speed: float
+ locationBearing: float
+ newDistanceLine: float
+ alpha360: float
+ width: float
+ height: float
+ midX: float
+ midY: float
+ xNew: float
+ yNew: float
+ xOld: float
+ yOld: float
+ distance: float
+ addedDistance: float
+ newDistance: float
+ alpha180: float
+ xWert: double
+ yWert: double
+ Himmelsrichtungen: Sting[]
- TWO_MINUTES: int
- MILLISECONDS_PER_SECOND: int
- UPDATE_INTERVAL_IN_SECONDS: int
- UPDATE_INTERVAL: long
- FASTEST_INTERVAL_IN_SECONDS:int
- FASTEST_INTERVAL: long
+ y: int
+ linien: List
+ floatArray: float[]
+ indices: short[]
+ degree()
+ Himmelsrichtung()
+checkForGoogleService()
+ createNewLocationRequest()
# isBetterLocation()
+ onLocationChanged()
+ getVertices()
+ getIndices()

MainActivity

Die MainActivity ist unsere Hauptklasse. Sie erbt von der Klasse Activity.
In der onCreate Methode, die aufgerufen wird, wenn die Activity das erstellt wird. Hier initialisieren wir die LocationGPS und surfaceView Objekte. Ebenfalls wird hier ein neues Layout erstellt und ihm die SurfaceView hinzugefügt.
In der onStart Methode stellen wir eine Verbindung zum Location Client her.Die onStart Methode wird immer aufgerufen sobald die Application wieder in den Vordergrund tritt. Tritt sie in den Hintergrund (onStop()), unterbrechen wir die Verbindung zum Client um Energie zu sparen.

```
public class MainActivity extends Activity {

    //Variablen

    public static TextView locationView;
    public static GLSurfaceView glSurfaceView;
    public LocationGPS locationGPS;



    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //Object der Klasse LocationGPS erstellen
        locationGPS = new LocationGPS(this);


        // erstellen eines Surface Views
        glSurfaceView = new GLSurfaceV(this);


        RelativeLayout layout = (RelativeLayout) findViewById(R.id.layout);

        RelativeLayout.LayoutParams glParams =
                new RelativeLayout.LayoutParams
                (RelativeLayout.LayoutParams.MATCH_PARENT,
                        RelativeLayout.LayoutParams.MATCH_PARENT);

        layout.addView(glSurfaceView, glParams);

        locationView = new TextView(this);
        locationView.setText("Herzlich Wilkommen bei Streck O Graph ");
        locationView.setTextColor(Color.parseColor("#00FF00"));
        locationView.setGravity(0x01);

        layout.addView(locationView);

    }
```

GLSurfaceV

Sie erbt von der Klasse GLSurfaceView. Sie stellt die Oberfläche dar, auf der gezeichnet wird. Im Konstruktor legen wir fest das wir OpenGL Version 2 verwenden und fügen einen Renderer hinzu, der fürs rendern der Grafik zuständig ist. Zusätzlich legen wir noch fest das nur Gerendert werden soll wenn sich was an den zu zeichnenden Daten ändert. Als letztes haben wir noch eine Methode onTouchEvent hinzugefügt. Sie dient zur Erkennung von Touch Gesten auf dem Display und schickt die Daten an die Renderklasse wo sie weiterverarbeitet werden.

```
public class GLSurfaceV extends GLSurfaceView {


private final GLRenderer mRenderer;

    public GLSurfaceV(Context context) {
        super(context);

        //OpenGl 2.0 Context erstellen
        setEGLContextClientVersion(2);

        // Renderer zum zeichnen erstellen
        mRenderer = new GLRenderer(context);
        setRenderer(mRenderer);

        // Nur zeichnen wenn sich die Daten geändert haben
        setRenderMode(GLSurfaceView.RENDERMODE_WHEN_DIRTY);
    }


    // Touch Event erstellen
    public boolean onTouchEvent(MotionEvent event) {
        return mRenderer.onTouchEvent(event);
    }
```

GLShader

Hier werden die beiden Shader erstellt. Der Vertex-Shader und der Fragment Shader. Der Fragment Shader legt die Farbe der einzelnen zu zeichnenden pixel fest. Hier 100% Grün. Der Vertex-Shader übersetzt die Modelspace Koordinaten in die Bildschirmkoordinaten. Mit der Methode loadShader() wird der gewünschte Shader geladen. Sie findet in der Render Klasse Verwendung.

```
public class GLShader {

    //Variablen
    public static int sp_SolidColor;



    public static final String vs_SolidColor =
        "uniform    mat4        uMVPMatrix;" +
        "attribute  vec4        vPosition;" +
        "void main() {" +
        "  gl_Position = uMVPMatrix * vPosition;" +

        "}";



    public static final String fs_SolidColor =
        "precision mediump float;" +
        "void main() {" +
        "  gl_FragColor = vec4(0,1,0,1);" +
        "}";




    public static int loadShader(int type, String shaderCode){

        // create a vertex shader type (GLES20.GL_VERTEX_SHADER)
        // or a fragment shader type (GLES20.GL_FRAGMENT_SHADER)
        int shader = GLES20.glCreateShader(type);

        // add the source code to the shader and compile it
        GLES20.glShaderSource(shader, shaderCode);
        GLES20.glCompileShader(shader);

        // return the shader
        return shader;
    }
}
```

GLRenderer

Sie ist für sämtliche Render Aufgaben verantwortlich und zeichnet letztendlich die Koordinaten auf die Surface View. In der SetupLines Methode wird das Array mit den zu zeichnenden Koordinaten und ein Array was die Reihenfolge der abzuarbeitenden Punkte beinhaltet erstellt, bzw. von der LocationGPS Klasse übermittelt. Hier werden auch die ByteBuffer erstellt, den wird die zuzeichnenden Daten übergeben.

```
public void SetupLines()
{

    floatArray = LocationGPS.getVertices();
    indices = LocationGPS.getIndices();


    // The vertex buffer.
    ByteBuffer bb = ByteBuffer.allocateDirect(floatArray.length * 4);
    bb.order(ByteOrder.nativeOrder());
    vertexBuffer = bb.asFloatBuffer();
    vertexBuffer.put(floatArray);
    vertexBuffer.position(0);

    // initialize byte buffer for the draw list
    ByteBuffer dlb = ByteBuffer.allocateDirect(indices.length * 2);
    dlb.order(ByteOrder.nativeOrder());
    drawListBuffer = dlb.asShortBuffer();
    drawListBuffer.put(indices);
    drawListBuffer.position(0);


}
```

Nach der erstmaligen erstellen der Surface View wird die onSurfaceCreated Methode aufgerufen. Hier wird erstmals die vorhandenen koordinaten Initialisiert. Die zu Anfangs allerdings noch leer sind. Der Hintergrund wird auf Schwartz gesetzt. Danach wird ein Shader Program erstellt und ihm die beiden Shader hinzugefügt.

```
@Override
public void onSurfaceCreated(GL10 gl, EGLConfig config) {

    // Create the triangle
    SetupLines();

    // Set the clear color to black
    GLES20.glClearColor(0.0f, 0.0f, 0.0f, 1);

    // Create the shaders
    int vertexShader = GLShader.loadShader(GLES20.GL_VERTEX_SHADER, GLShader.vs_SolidColor);
    int fragmentShader = GLShader.loadShader(GLES20.GL_FRAGMENT_SHADER, GLShader.fs_SolidColor);

    GLShader.sp_SolidColor = GLES20.glCreateProgram();             // create empty OpenGL ES Program
    GLES20.glAttachShader(GLShader.sp_SolidColor, vertexShader);   // add the vertex shader to program
    GLES20.glAttachShader(GLShader.sp_SolidColor, fragmentShader); // add the fragment shader to program
    GLES20.glLinkProgram(GLShader.sp_SolidColor);                  // creates OpenGL ES program executables

    // Set our shader programm
    GLES20.glUseProgram(GLShader.sp_SolidColor);
}
```

In der onSurfaceChanged werden Projektions und View Matrizen erstellt und unser Viewpoint gesetzt

.

```
@Override
public void onSurfaceChanged(GL10 gl, int width, int height) {

    // We need to know the current width and height.
    mScreenWidth = width;
    mScreenHeight = height;

    // Redo the Viewport, making it fullscreen.
    GLES20.glViewport((int)(0+mAngleX), (int)(0 + mAngleY), (int)(mScreenWidth+mAngleX),

    // Clear our matrices
    for(int i=0;i<16;i++)
    {
        mtrxProjection[i] = 0.0f;
        mtrxView[i] = 0.0f;
        mtrxProjectionAndView[i] = 0.0f;
    }

    // Setup our screen width and height for normal sprite translation.
    Matrix.orthoM(mtrxProjection, 0, 0f, mScreenWidth, 0.0f, mScreenHeight, 0, 50);

    // Set the camera position (View matrix)
    Matrix.setLookAtM(mtrxView, 0, 0f, 0f, 1f, 0f, 0f, 0f, 0f, 1.0f, 0.0f);

    // Calculate the projection and view transformation
    Matrix.multiplyMM(mtrxProjectionAndView, 0, mtrxProjection, 0, mtrxView, 0);

}
```

Die Daten aus den ByteBuffern werden in der Render Methode weiterverarbeitet, wo auch der Zeichenaufruf stattfindet.

```
private void Render(float[] m) {

    // clear Screen and Depth Buffer, we have set the clear color as black.
    GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT | GLES20.GL_DEPTH_BUFFER_BIT);

    // get handle to vertex shader's vPosition member
    int mPositionHandle = GLES20.glGetAttribLocation(GLShader.sp_SolidColor, "vPosition");

    // Enable generic vertex attribute array
    GLES20.glEnableVertexAttribArray(mPositionHandle);

    // Prepare the Line coordinate data
    GLES20.glVertexAttribPointer(mPositionHandle, 3,GLES20.GL_FLOAT, false,0, vertexBuffer);

    // Get handle to shape's transformation matrix
    int mtrxhandle = GLES20.glGetUniformLocation(GLShader.sp_SolidColor, "uMVPMatrix");

    // Apply the projection and view transformation
    GLES20.glUniformMatrix4fv(mtrxhandle, 1, false, m, 0);

    GLES20.glLineWidth(5.0f);


    GLES20.glDrawElements(GLES20.GL_LINE_STRIP, indices.length, GLES20.GL_UNSIGNED_SHORT, drawListBuffer);


    // Disable vertex array
    GLES20.glDisableVertexAttribArray(mPositionHandle);

}
```

Die Render Methode wird in der onDrawFrame Methode aufgerufen. OnDrawFrame wird einmalig nach erstellen des Renderes aufgerufen und jedes mal nach dem Render Request. Sie ist quasi der Aufruf zum Zeichnen.

```
@Override
public void onDrawFrame(GL10 unused) {

    //Toast.makeText(mContext, "onDrawFram"+i, Toast.LENGTH_
    // Get the current time
    long now = System.currentTimeMillis();
    GLES20.glViewport((int)(0+mAngleX), (int)(0 - mAngleY),

    // We should make sure we are valid and sane
    if (mLastTime > now) return;

    // Get the amount of time the last frame took.
    //long elapsed = now - mLastTime;


    // Update our example
    SetupLines();
    // Render our example
    Render(mtrxProjectionAndView);

    // Save the current time to see how long it took :).
    mLastTime = now;

}
```

Am Ende der Klasse werden in der onTouch Methode die Daten aus der Touchgeste verarbeitet und damit der Viewpoint auf die Zeichenfläche verändert.

```
public boolean onTouchEvent(MotionEvent e) {
    float x = e.getX();
    float y = e.getY();

    switch (e.getAction()) {
    case MotionEvent.ACTION_MOVE:
        dx = x - mPreviousX;
        dy = y - mPreviousY;
        mAngleX = (mAngleX + (int)(dx * TOUCH_SCALE_FACTOR));
        mAngleY = (mAngleY + (int)(dy * TOUCH_SCALE_FACTOR)) ;

        break;
    }
    mPreviousX = x;
    mPreviousY = y;
    //Toast.makeText(mContext, "Touch Event", Toast.LENGTH_SHORT).show();
    MainActivity.glSurfaceView.requestRender();
    return true;
}
```

Die LocationGPS Klasser implementiert vom GooglePlayServicesClient die Connection Callbacks und den onConnect Failed Listender. Zusätzlich wird noch ein Location Listener implementiert.
Im Konstruktor der Klasse wird als erstes die Verfügbarkeit des GooglePlay Services abgefragt, da der für die Standortdaten notwendig ist. Es werden Location Client und ein Location Request erstellt. So lange die Verbindung zum Client aufgebaut ist werd Standortdaten im 5 Sekunden bis maximal 1 Sekunden Tackt angefordert. Ebenfalls wird hier einmalig die Bildschirmauflösung abgefragt.

```
public LocationGPS(Context context) {
    // TODO Auto-generated constructor stub
    this.context = context;

    //Abfrage ob Google Play Service verfügbar ist
    checkForGoogleService();

    //Location Client Objekt erstellen
    mLocationClient =  new LocationClient(context, this, this);

    //neuen LocationRequest erstellen
    createNewLocationRequest(LocationRequest.PRIORITY_HIGH_ACCURACY,

    //Display Auflösung abfragen
    WindowManager wm = (WindowManager) context.getSystemService(Cont
    Display display = wm.getDefaultDisplay();

    Point size = new Point();
    display.getSize(size);
     width = size.x/2;
     height = size.y/2;
     xOld = width;
     yOld = height;

}
```

Die onLocationChanged Mehtode ist die wichtigste in unserer Anwendung. Ihr werden die neuen Location Objekte übergeben. Hier ermitteln wir die Abständer der einzelnen Locations, sowie ihre Geschwindigkeit und Richtung. Diese Daten sind bereits im Location Objekt gespeichert. Daraus berechnen wir mittels Trigonometrie die benötigten X und Y werte zum zeichnen der Wegstrecke und fügen diese einer ArrayListe zu. Danch wird ein neuer Render Aufruf gestartet.

```
@Override
public void onLocationChanged(Location location) {
    // TODO Auto-generated method stub


            //Überprüpft ob die neue Location genauer ist als die alte
            if(isBetterLocation(location,currentBestLocation)){
                currentBestLocationOld = currentBestLocation;
                 currentBestLocation = location;
            //setzt die erste empfangene Location als anfangsLocation
            if(anfangsLocation == null){
                 anfangsLocation =  currentBestLocation;
                }


            try{

            //Richtung vom alten Standort zum neuen (von -180° bis +180°)
            alpha180 = currentBestLocationOld.bearingTo(currentBestLocation);
            //Abstand der beiden Standorte
            newDistanceLine = currentBestLocationOld.distanceTo(currentBestLocation);
            //Berechnung der Gesamtstrecke
            addedDistance = addedDistance + newDistance;
            //Berechnung der Geschwindigkeit in km/h
            speed = (int)Math.round((currentBestLocation.getSpeed())*3.6F);
            //Umwandlung der 180° Gradwertes ind 360° Gradwert
            alpha360 = degree(alpha180);
```

```
}
//Unwandlung des Gradwertes in Radiant
double alpha360Rad =  Math.toRadians(alpha360);
//Kosinus und Sinus der Khateten ausrechnen
double cosA = Math.cos(alpha360Rad);
double sinA = Math.sin(alpha360Rad);
//Die neuen X und Y Werte berechnen | Die Entfernung des neuen Pun
// Mittels der Trigonometrie die Länger auf der X und der Y Achse
float xValue = ((newDistanceLine * (float)sinA)*30);
float yValue = ((newDistanceLine * (float)cosA)*30);
//Richtung im Bezug auf den Startpunkt ermitteln
locationBearing = anfangsLocation.bearingTo(currentBestLocation);
himmelsrichtungenIndex = Himmelsrichtung(locationBearing);

String text = "Speed: "+speed+" km/h    |  "+Himmelsrichtungen[himm



MainActivity.locationView.setText(text);


xNew = (xOld + xValue);
yNew = (yOld + yValue);

linien.add(xOld);
linien.add(yOld);
linien.add(0.0f);

xOld = xNew;
yOld = yNew;



y = y +1;
indices = new short[y];
floatArray = new float[linien.size()];

for(int x = 0; x <indices.length;x++){
    indices[x] = (short) x;

}


int p = 0;

for (Float f : linien) {
    floatArray[p++] = f; // Or whatever default you want.
}




MainActivity.glSurfaceView.requestRender();
```

3. REFLEXION

Rückblickend war es eine interessante Erfahrung im Team eine App zu entwickeln, da es das erste Mal für uns war. Ein bisschen unglücklich fanden wir zwar, dass die Projektarbeit von anfangs drei Mitgliedern dann auf einmal auf zwei beschränkt wurde und wir uns nur dadurch zusammenschlossen, weil wir jeweils die dritte Person waren aber wir waren uns schnell einig, welche Art von App wir entwickeln wollten. Beim technischen Teil wurden wir zügig damit konfrontiert, dass wir uns Zuviel vorgenommen hatten und die Umsetzung vereinfachen mussten. Als die Funktionen soweit standen war das nächste größere Problem die Zeichnung des zurückgelegten Weges. Diesbezüglich mussten wir nochmals viel Aufwand aufbringen wodurch das Design ein wenig in den Hintergrund geriet.

Der Tag der Präsentation war eine gute Gelegenheit sich in der Wartezeit mit den anderen Projekten auseinanderzusetzen und auszuprobieren.