Projection.h

<Pxl> bildet Elemente von <zBuffer> ab. <Coord> bildet Elemente von <facePixelBuffer> ab.

«struct»
Pxl

z:float
c:Color

«struct»
Coord

hor:int
ver:int
bool operator* <(coord& c{read only})

Um den Vergleichoperator[<] für Werte dieser Struktur zu redefinieren

In der Simulation wird die virtuelle Welt um die Kamera transformiert. Daraus resultiert dass die Achsen des Koordinatenkreuz', auf welche bei Rotation-Tranformationen referenziert wird, veränderbar existieren müssen.

Projection

+zBuffer : std::vector<std::vector<Pxl> >
+camPos: std::vector<float>
+transMat: std::vector<float>
+projMat: std::vector<float>
-transformation:Transformation*
-facePixelBuffer: std::set<coord>
-absCoordA: std::vector<float>
-absCoordB: std::vector<float>
-mappedPointA: std::vector<float>
-mappedPointB: std::vector<float>
-coordCrossX: std::vector<float>
-coordCrossY: std::vector<float>
-coordCrossZ: std::vector<float>

-bresenline (ax:int, ay:int, bx:int, by:int, pntAzwei:float {read only},
SUBy1y2:float {read only}) :void
-rasterization (c:Color*{read only}) :void
+Projection (argc:int, argv[]:char*, transformationArg:Transformation*)
+isVisible (faceId:unsigned int{read only}, current:Model*{read only}) :bool
+map(current:Model*) :void
+getCoordCrossX: *std::vector<float>
+getCoordCrossY: *std::vector<float>
+getCoordCrossZ: *std::vector<float>
+setCoordCrossX(v: std::vector<float>): void
+setCoordCrossY(v: std::vector<float>): void
+setCoordCrossZ(v: std::vector<float>): void

Reader.h

Reader

-fs: std::fstream
-s: std::string
-name: std::string
-i,j: int
-amtPnts: unsigned int
-amtFaces: unsigned int
-pointsPerFace: std::vector<unsigned int>
-alignment: std::vector<std::vector<float> >
-pos: std::vector<float>
-points: std::vector<std::vector<float> >
-faces: std::vector<std::vector<unsigned short> >

-allocPntrs(): void
+readData(modelList:std::vector<Model>*{read only},
colorList:std::vector<struct Color>*{read only},
transformation: Transformation*): void
+readAnim(animList:std::vector<Anim>*{read only},
modelList: std::vector<Model>*{read only},
transformation: Transformation*): void
+readTrafo(animList:std::vector<Anim>*{read only},
transformation: Transformation*): void

Target.h

Target

-totalNrSteps: int
-stepsDone: int
-from: Model*
-to: Model*

+Target (fromArg: Model*,toArg: Model*)
+getTotalNrSteps(): int
+getStepsDone(): int
+resetStepsDone(): void
+incrementStepsDone(): void
+getFrom(): Model*
+getTo(): Model*

Anim.h

Anim

-m: Model*
-targetPointingOn: int
-direction: char
-targets: std::vector<Target*>

+Anim (m: Model*)
+getNrTargets(): int
+focusNextTarget(): void
+getDirection(): char
+getModel():Model*
+addTarget(newTarget: Model*)
+getTarget(i: int): Target*
+focussedTarget(): Target*

Transformation.h

Transformation

-camPos: std::vector<float>*
-std::vector<float> moveDirectionX={1,0,0,0}
-std::vector<float> moveDirectionY={0,1,0,0}
-std::vector<float> moveDirectionZ={0,0,1,0}

+move (m:Model*{read only},
direction:char{read only}
speed:int{read only}) :void
+rotatePoint (alpha: float{read only},axis: char{read only},
pnt:std::vector<float>*{read only},
ret:std::vector<float>*{read only}): void
+rotateFigure(alpha: float{read only},axis: char{read only},
m:*Model{read only}): void
+rotate (alpha: float{read only},axis: char{read only},
m:*Model{read only}): void
+rotate (alpha: float{read only},axis: char{read only},
m:*Model{read only},um:std::vector<float>*): void
+matMul (mA:std::vector<float>*{read only},
rowsA: unsigned int{read only},
colsB: unsigned int{read only},
clsArwsB: unsigned int{read only},
mB: std::vector<float>*{read only},
mRet: std::vector<float>*{read only}): void
+align (m:Model*{read only},target:std::vector<float>target):void
+rotateCam (alpha:float{read only},
axis: char{read only},
modelListArg:std::vector<Model>*,
animListArg:std::vector<Anim>*,
camPos: std::vector<float>*{read only}): void
+changeAlignment (m:Model*{read only},target:Model*{read only})
+moveCamAlongAlignment (modelListArg:std::vector<Model>*,
direction:char{read only}): void

Start.cpp

projection: Projection*
transformation: Transformation*
modelList: std::vector<Model>
animList: std::vector<Anim>
colorList: std::vector<struct Color>
isFS:bool=false

main (argc:int,argv[]:char*): int
keyboardFunc (key:unsigned char,x:int,y:int):void
displayCallback():void

Schnittstelle zum Grafik- und Steuerungs-Api.
Um einzelnes Pixel zu färben und Funktionsaufrufe Tasten zuzuweisen.
In main werden Api-spezifische initialisierungen vorgenommen und das System wird gestartet.

Model.h

«struct»
Color

isColored:bool
red:float
green:float
blue:float

Model

-faces: std::vector<std::vector<unsigned short> >
-points: std::vector<std::vector<float> >
-alignment: std::vector<std::vector<float> >
-getsProjected
+pos: std::vector<float>
+color: Color*
+name:std::string

-extractPntIDsOf(mPart: *Model): void
-addJointsAxis(nArg: std::vector<float>{read only}): void
+Model (nameArg:std::string, posArgstd::vector<float>,
pointsArg:std::vector<std::vector<float> >,
facesArg: std::vector<std::vector<unsigned short> >)
+~Model() :void
+getPoints(): std::vector<std::vector<float»*
+getPoint(id: unsigned int): std::vector<float>*
+getPos(): std::vector<float>*
+getAlignmentsLineVec(x: unsigned int): std::vector<float>*
+getFace(faceId: unsigned int): std::vector<unsigned short>
+direction: char
+getFacesSize(faceId: unsigned int):unsigned int
+getFaceSize(): unsigned int
+getPointsCoord(i: unsigned short,c: unsigned short): float
+getPointsSize(): unsigned int
+getAlignment(x: unsigned int,y:unsigned int): float
+setPoint(i: unsigned int,pnt: std::vector<float>): void
+setAlignment(I: unsigned int,align:std::vector<float>): void
+addJoint(mPart: *Model,nArg: std::vector<float>{read only}): void
+rotateJoint(alpha: float{read only}, id: unsigned int{read only}): void
+getJoints(): std::vector<std::vector<float»
+setJoint(id: unsigned int, nArg: std::vector<float>): void
+getGetsProjected(): bool
+setGetsProjected(b: bool{read only}) void