CamerAH

eine OpenGL-Kamera mit ArcBall-Rotation

Downloadseite

Beschreibung :

glCamerAH ist eine freie OpenGL-Kamera nach MPL-Lizenz, die einfach in Projekte integriert werden kann.
Von der im Wiki beschriebenen Kamera von Andree inspiriert, habe ich eine solche nach einem etwas anderen Konzept neu aufgesetzt. Insbesondere findet die Unit VectorGeometry aus GLScene weite Anwendung. Die ModelView-Matrix wird als Cameramatrix in glCamerAH geführt und verarbeitet. Neben der Cameraposition (PointofCamera) und dem Viewpunkt (PointofView) gibt es ein Rotationspunkt (PointofRotation), um den gedreht wird. Der Punkt ist unabgängig und kann verschoben werden.
Optional kann der Perspectiv- und Ortho-Modus von der Kamera verwaltet werden. Dadurch wird es möglich, den Fensterzoom (+/-) zu nutzen.
Ebenfalls ist eine ArcBall-Rotation eingefügt.

Entwickelt wurde die Kamera mit Delphi 2009, andere Delphi-/Pascal-Versionen wurden nicht getestet. Sollte es dort nicht laufen, bitte ich um Nachricht, um es auch für die Versionen anpassen zu können.
Es werden die Fremdkomponenten VectorTypes und VectorGeometry der "neuen" GLScene genutzt, die aber im Zip enthalten sind. Also GLScene selbst o.ä. muß nicht vorhanden sein. (Allerdings muß selbstverständlich dglopengl zugänglich sein.)

Features :

- Rotation um X und Y-Achse
- Rotation um die einzelnen Achsen
- Rotationsspeed
- Move in X und Y-Richtung und in die Tiefe
- Movespeed
- Verschieben des Rotationpunktes (oder nicht)
- Verwalten der Perspektiv-Modi
- Ausgleich der Verzerrung (Höhe, Breite des Fensters) z.B. beim Resize
- mehrfacher Fensterzoom (+/-)
- Setzen der Ansichten auf XY, XZ, ...
- ArcBall-Rotation

Schnellstart :

```
Camera: TglCamerAH;
pnl1: TPanel;

procedure TForm1.FormCreate(Sender: TObject);
begin
  DefContext; // ← (InitOpenGL, CreateRenderingContext, ActivateRenderingContext)

  Camera := TglCamerAH.Create;

  Camera.PositionCamera(
   VectorMake(0.5,0.5, -1),
   VectorMake(0.5, 0.5, 0.5),
   VectorMake(0,1,0),
   VectorMake(0.5, 0.5, 0.5));

 Camera.SetProjection(-1, 1,  -1, 1,  1.0, 5);
  // optional:
 Camera.OGLControl := pnl1;
end;
```

Create;
Die Camera wird als Camera: TglCamerAH deklariert. Und in FormCreate des Fensters wird die Camera dann created. Das Create muß nach dem Initialisieren von OpenGL(InitOpenGL, CreateRenderingContext, ActivateRenderingContext) erfolgen.
Die Camera muß selbstverständlich in Formdestroy wieder mit Free befreit werden.

PositionCamera(Camera, Center, Up, PoR: TVector);
Dann folgt die Positionierung der Camera. Das ist bis auf den letzten Parameter gleich mit gluLookat. Der letzte Parameter PoR ist die Position des Rotationpunktes. Camera ist die Position des Kamera, Center ist der Punkt auf den die Kamera ausgereichtet ist und Up ist der Vektor für den nach oben ausgerichteten Achse.

SetProjection(left, right, bottom, top, zNear, zFar: Single);
Hier wird die Projektion eingestellt. Die Parameter entsprechen denen von glFrustum und glOrtho, die entsprechend in der Camera, je nach Einstellung der Perspektive, damit gesetz werden.

OGLControl
OglControl muß mit dem TControl gesetzt werden, worauf gerendert wird. Üblicherweise ist das wie hier TPanel oder auch Form. Das wird benötigt, um den Cursor mit den internen Icons zu setzen.

```
procedure TForm1.Render;
begin
  glClearColor(0.3, 0.4, 0.7, 0.0);    //Hintergrundfarbe: Hier ein leichtes Blau
  glEnable(GL_DEPTH_TEST);      //Tiefentest aktivieren
  glEnable(GL_CULL_FACE);       //Backface Culling aktivieren

  glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);

  Camera.Projection(pjPerspectiv)
  //Camera.Projection(pjOrtho); // alternativ;

  Camera.Apply;

  glPushMatrix;
    glTranslatef(0.25, 0.25, 0.25);
    glColor4f(1, 0, 0, 0.7);
    Cube(0.5, 0.5, 0.5, false, voffset, toffset, tscale); // zeichnet einen Würfel
  glPopMatrix;

  SwapBuffers(DC);
end;
```

Projection(PMode: TProjection);
Projektion muß zu Beginn jeden Renderdurchlauf gesetzt werden. Intern wird dann damit optimiert glOrtho bzw glFrustum aufgerufen. Der Parameter **pjPerspectiv** steht für die perspektive Darstellung und **pjOrtho** für die orthogonale.

Apply;
Das entspricht dem Aufruf :
```
glMatrixMode(GL_PROJECTION);
glLoadIdentity;
```
Überall dort, wo dieser Aufrufe notwendig sind, muß jetzt **Apply** stehen.

ApplyInvers;
Zusätzlich gibt es **ApplyInvers**. Das kann dazu genutzt werden, um Objekte, unabhängig von der Drehung, immer zur Kamera auszurichten. Ein Beispiel wäre Schrift, die dann immer so ausgerichtet ist, dass sie lesbar ist.

```
procedure TForm1.pnl1MouseDown(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then
  begin
    Camera.RotateStart(X, Y);
    Render;
  end;
end;

procedure TForm1.pnl1MouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);
begin
  if Camera.RotateMove(X, Y) then
    Render;
end;

procedure TForm1.pnl1MouseUp(Sender: TObject; Button: TMouseButton; Shift:
TShiftState; X,  Y: Integer);
begin
  Camera.RotateStop;

  if Camera.IsafterRotation and (Camera.RotMode = rmArcBall) then
    while Camera.afterRotation do
    begin
      if GetAsyncKeyState(VK_LBUTTON) < 0 then
        Exit;
      Render;
    end;
  Render;
end;
```

RotateStart(X, Y: Integer);
RotateMove(X, Y: Integer);
RotateStop;
RotateStart muß im MouseDown-Event stehen, unabhängig von der Bewegungsart (Rotate, Move ArcBall...). **RotateMove** im MouseMove-Event und **RotateStop** im MouseUp-Event.
Die Parameter X, Y sind immer die X, Y Variablen aus den Events.
Nach den Aufrufen muß immer ein Renderdurchlauf folgen.

AfterRotation;
Die While-Schleife mit AfterRotation wird nur benötigt, wenn IsafterRotation auf True gesetzt ist. AfterRotation bewirkt, dass dem ArcBall ein gewisser Schwung versetzt werden kann und der Ball nach dem Loslassen weiter läuft. So wie in Google-Earth.
GetAsyncKeyState(VK_LBUTTON) fragt die linke Maustaste ab, um bei Betätigung auszusteigen und die Nachrotation zu stoppen.

```
procedure TForm1.pnl1Resize(Sender: TObject);
begin
  if Assigned(Camera) then
  begin
    glViewport(0,0, pnl1.Width, pnl1.Height);
    Camera.Resize;
    Render;
  end;
end;
```

Resize;
Das **Resize** der Camera muß bei Veränderung des Renderuntergrundes (TPanel oder TForm) ausgelöst werden. Danach muß ein Renderdurchlauf erfolgen.
Da die Resize-Events schon vor der Deklaration der Camera feuern, muß mit Assigned zurvor abgefragt werden, ob die Camera schon created wurde.

```
procedure TForm1.btn1Click(Sender: TObject);
begin
  Camera.RotMode := rmFlexAxis;
end;
```

Camera.RotMode
Mit dem **RotMode** wird das Verhalten der Kamera eingestellt, welches bei Betätigung der Maustasten ausführt.
Folgende Einstellung sind derzeit möglich:
rmFlexAxis
dreht gleichzeitig um starre vertikale und horizontale Achse

rmArcBall
dreht einen imginären Ball

rmMove
schiebt gleichzeitig nach links und rechts

rmDepthMove
schiebt in die Tiefe

rmRotX, rmRotY, rmRotZ
dreht jeweils um die starren Achsen

rmRotCoordX, rmRotCoordY, rmRotCoordZ
dreht jeweils um die imaginäre Achse der Szene

rmZoomPlus, rmZoomMinus
zieht ein Fenster auf und setzt den Ausschnitt auf die gesamte Bildfläche

rmNothing
tut nichts

rmSelect
wählt Objekte mit Hilfe der PickMatrix aus

rmColorPick
gibt die Farbe unter dem gedrückten Mauscursor zurück. U.a. für einfaches ColorPicking gedacht