PGA CALCULATIONS

REPORT

Ivans Minajevs ivans.minajevs@student.howest.be 2GD11

TWO-BLADE – TWO-BLADE INTERSECTION

My race game includes map, that consists of two blades (borders). I loop through all of them in order to check whether one of the sides (2-blades) of Player car intersects with the wall. (Collision checking)

L – player side two blade, B – map border two blade

1) Skew test: $L \wedge B = 0$. In case two lines are not skew they are linearly dependent, have a common point & plane.
2) Calculation:

$$P = ((\langle LB \rangle_2 \wedge e_0) \vee L) \wedge B$$

where P – the common point between two two-blades, $\langle LB \rangle_2$ – a common normal (2-blade, grade 2 of line-line gep), $((\langle LB \rangle_2 \wedge e_0) \vee L)$ – a common plane.

ORBITING AROUND POINT IN CERTAIN DIRECTION

In my game when clicking on screen the car orbits left or right depending on its current forward two-blade.

OP – 2-blade from origin to orbit point (click location 3-blade), F – forward 2-blade that stores values in vanishing part, representing car look-at direction, ω – scalar, angular velocity, CP – 2-blade from car centre to orbit point.

1) Commutator product between plane norm and forward direction in order to find line perpendicular to forwards two-blade (aka side direction 2-blade)

$$S = \langle \hat{n}\, D(F) \rangle_2$$

where $\hat{n}$ – normal of the plane formed by screen, $D(F)$ – dualized forward 2-blade, to perform calculations on Euclidean part, S – grade 2 of line-line gep, commutator product.

2) Calculate dot product between car side direction and CP

$$t = S \cdot CP$$

3) Determine the rotation line *I* based on value of t. Case $t > 0$ rotation line is pointing outwards (from the camera), hence making sure car orbit right, case $t \leq 0$ rotation line points into the camera, allowing rotation to the left.
4) Create an origin rotator $R_{O\omega I}$

5) Create an off-origin rotator, having $T_{\overline{OP}}$ translator in the direction of OP of the distance of the norm of OP

$$R_{\omega \mathrm{I}} = T_{\overline{OP}} R_{\mathrm{O}\omega \mathrm{I}} {T_{\overline{OP}}}^{-1}$$

6) For each car point (x_n) apply an off-origin rotation

$$x_n' = R_{\omega \mathrm{I}} x_n {R_{\omega \mathrm{I}}}^{-1}$$

CAMERA

Since my game requires a large map for convenient controls and movement, I implemented Camera transform, allowing for larger 2D projects.

1) Calculate Aim
   1.1) Calculate camera X, Y positions, based on track center (P) 3-blade

$$cameraX = p1 - \frac{screenWidth}{2}$$
$$cameraY = p2 - \frac{screenHeight}{2}$$

   where $p1$ - point P vanishing part $e032$ coefficient, $p2$ - point P vanishing $e032$ coefficient

   1.2) Snap camera Y and X positions in case trying to go outside of the map borders.
   1.3) Create camera translation vector, defined with Plucker line as:

$$L = [(0,0,0);(-cameraX,-cameraY,0)]$$

   1.4) Create a final transform motor $T_{\overline{L}}$ of the distance of vanishing norm of line $|L|_\infty$ and store it
2) Get applied transform, given the user provided a 3-blade as input X.

$$X' = T_{\overline{L}} X {T_{\overline{L}}}^{-1}$$

3) Get world location of an object.
   To get the initial X before the transform I sandwich X' with $T_{\overline{L}}$ in reverse order

$$X = {T_{\overline{L}}}^{-1} X' T_{\overline{L}}$$

   Proof:

$$T_{\overline{L}}^{-1} X' T_{\overline{L}} = \underbrace{T_{\overline{L}}^{-1} T_{\overline{L}}}_{1} X \underbrace{T_{\overline{L}}^{-1} T_{\overline{L}}}_{1} = X$$