

Collision Motion Model Design

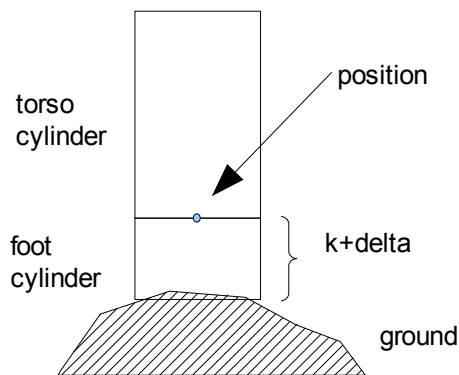
Chris Darken and Brad Anderegg

Version 1.0

September 27, 2006

This is a design for a first-person shooter-like motion model that respects a polygonal mesh called the “collision geometry”. The desired characteristics of this model in actual use are:

- No “shake”
- Slides smoothly along walls.
- Does not get stuck inside the walls or floor, including floors cluttered with small objects
- Does not allow climbing walls (including slightly tilted ones)



The geometry of this motion model consists of two cylinders, one representing the body from the knees up, and the other from the knees down to just below the surface stood on. (A box is a possible alternative, but we have gotten stuck sliding around corners with obtuse angles.) The upper cylinder is never to intersect any polygon in the collision geometry in any rendered frame, while the lower one routinely intersects the collision geometry when not falling. Cylinder sides are always to be vertical. For this description, the position of the upper cylinder is taken to be the center of its bottom face. The upper cylinder is placed at knee height to allow the model to move up (gentle) slopes. The lower cylinder is of height $k + \delta$, where δ is to extend below the surface stood on. It is assumed that the cylinder is initialized to a non-intersecting position in frame 1 of any run. The model has a key variable called the mode, which must be one of WALKING, FALLING, or SLIDING. The initial value for mode is FALLING. $bJumping$ is initialized to false. Velocities are all taken to be 3D vectors. The z component will only be used when falling. $terminalSpeed$ is the minimum value of the velocity's z component (i.e. it limits how negative it can be). $slideThreshold$ determines how steep a slope must be for the model to slide. $slideSpeed$ is a parameter of the model that determines how quickly it slides down steep slopes. $jumpSpeed$ is the initial vertical speed when jumping. The algorithm then proceeds as follows:

1. Let the current position of the cylinder be p_0
2. Let $oldMode = mode$
3. If $bJumping$ and $vLast.z \leq 0$, assign $bJumping = false$. *$bJumping$ is used to avoid immediately clamping to the surface in the frame after the user requests a jump. Remember, the lower cylinder is generally intersecting the geometry and at high frame rates may require more than one frame to clear it.*
4. If mode is WALKING, get requested velocity vector (result of user key presses), v_0 . If the

- jump key is pressed, set `bJumping=true`, and `v0.z` to `jumpSpeed`, otherwise let `v0.z = 0`.
5. If mode is SLIDING, set `v0` to `vSlide`
 6. If mode is FALLING, let `fallSpeed = vLast.z - gravityAccel * deltaFrameTimeSecs`. If `fallSpeed < terminalSpeed`, `fallSpeed = terminalSpeed`. Let `v0 = (vLast.x, vLast.y, fallSpeed)`
 7. Generate cylinder position `p1` where `p1 = p0 + v0 * deltaFrameTimeSecs`.
 8. `TestPosition(p1, deltaFrameTimeSecs)`, pseudocode below.
 9. If there are no collisions, `p1` (as modified by `TestPosition`) is accepted as the model's position for the next frame. Let `vLast = v0`.
 10. If there are collisions, harvest all intersecting polygons and compute their normals.
 11. Initialize `v1 = v0`
 12. For each intersecting polygon normal `n`, if `n*v1 < 0`, then `v1 = v1 - (n*v1)n`. (Star represents vector dot product, aka scalar product, aka inner product). Note that all vectors are to be 3D to handle grazing walls while falling.
 13. Let `p2 = p0 + v1 * deltaFrameTimeSecs`
 14. `TestPosition(p2, deltaFrameTimeSecs)`, pseudocode below
 15. If there are no collisions, `p2` (as modified by `TestPosition`) is accepted as the model's position for the next frame. Let `vLast = v1`.
 16. If there are collisions, `p0` is the model's position for the next frame (i.e. it stays where it was in the last frame). Let `vLast` be `(0, 0, 0)`. Set `mode = oldMode`.

`TestPosition(p, dt)`

Tests candidate position p. Sets newMode and vSlide in case the position is actually adopted.

1. Collide lower cylinder with collision mesh.
2. If there are no collisions or if `bJumping` is true, set `mode = FALLING`. If both are true, set `bJumping` false. *Note that we are "falling" even at the beginning of a jump.*
3. Otherwise, let `z` be the `z` component of the uppermost contact point. Let `z' = min(z, jumpSpeed * dt)`. Let `p = (p.x, p.y, z'+k)`.
 - a) Harvest all intersecting polygons and compute their normals.
 - b) Let `n` be the normal with the maximum value of `n.z`. If `n.z` is less than `slideThreshold`, set `newMode = SLIDING` and let `vSlide = (slideSpeed / (1-n.z)) * (n.z*n.x, n.z*n.y, n.z*n.z - 1)`. *vSlide is tangent to the surface and pointing downhill.* Otherwise set `newMode = WALKING`.
4. Move the cylinder to `p` and test for collision.

Known Problems

- You can get stuck in ravines with steep walls. You end up in SLIDING mode with no control.
- Can get shake trying to force your way up steep surfaces (e.g. walls of shell craters).
- Sliding has no inertia. You slide directly downhill even when this results in sharp changes of direction at polygon boundaries.