0456024 蔡佩珊

Computer Animation Project Assignment 1

♦ Topic

Forward Kinematics

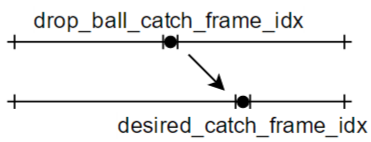
♦ Implementation (see commands in code)

ForwardSolver::ComputeSkeletonPose()

```
// R(a->b): rotation matrix from local_coord_a to local_coord_b
// R_asf(i): rotation matrix from local_coord_i to local_coord_i'parent
// R_amc(i): rotation matrix for i in local_coord_i at a particular frame
// unitV(i)/V(i): unit vector/vector from local_coord_i_origin to local_coord_i'child_origin
// L(i): length of i
// T(i): start poisition of i in global_coord
PoseColl_t ForwardSolver::ComputeSkeletonPose(const math::Vector6dColl_t &joint_spatial_pos)
{
    // R(i+1->i) = R_asf(i)*R_acm(i)
    math::RotMat3d_t *rot_cb2pb = new math::RotMat3d_t[skeleton_->bone_num()];
    for (int i = 0; i < skeleton_->bone_num(); ++i) {
        const acclaim::Bone* cb = skeleton_->bone_ptr(i);

        math::Vector3d_t amc_r = math::ToRadian(math::Vector3d_t(joint_spatial_pos[i][0], joint_spatial_pos[i][1], joint_spatial_pos[i][2]));
        math::RotMat3d_t amc_rot = math::ComputeRotMatXyz(amc_r[0], amc_r[1], amc_r[2]);
        rot_cb2pb[i] = math::ToRotMat(cb->rot_parent_current).transpose()*amc_rot;
    }

    PoseColl_t fk_pose(skeleton_->bone_num());
    // root
    math::Vector3d_t root_pos = math::Vector3d_t(joint_spatial_pos[0][3], joint_spatial_pos[0][4], joint_spatial_pos[0][5]);
    fk_pose[0].set_start_pos(root_pos);
    fk_pose[0].set_end_pos(fk_pose[0].start_pos());
    fk_pose[0].set_rotation(rot_cb2pb[0]);

    for (int i = 1; i < skeleton_->bone_num(); ++i) {
        const acclaim::Bone* cb = skeleton_->bone_ptr(i);
        const acclaim::Bone* pb = cb->parent;
        // R(i-1->g) = R(0->g)*R(1->0)*R(2->1)*...*R(i-1->i-2)
        math::RotMat3d_t rot_pb2global = fk_pose[pb->idx].rotation();
        // V(i-1) = unitV(i-1)*L(i-1)
        math::Vector3d_t dir_pb = pb->dir*pb->length;
        // T(i-1)
        math::Vector3d_t startpos_pb = fk_pose[pb->idx].start_pos();
        // T(i) = R(i-1->g)*V(i-1) + T(i-1)
        fk_pose[i].set_start_pos(rot_pb2global*dir_pb + startpos_pb);
        // R(i->g) = R(0->g)*R(1->0)*R(2->1)*...*R(i-1->i-2)*R(i->i-1) = R(i-1->g)*R(i->i-1)
        fk_pose[i].set_rotation(rot_pb2global*rot_cb2pb[i]);
        // T(i).end = T(i+1) = R(i+1-1->g)*V(i+1-1) + T(i+1-1) = R(i->g)*V(i) + T(i)
        fk_pose[i].set_end_pos(fk_pose[i].rotation()*(cb->dir*cb->length) + fk_pose[i].start_pos());
    }

    return fk_pose;
}
```

TimeWarper::ComputeWarpedMotion()

重新定義 hard constraint(frame_idx,play_second)：在時間點 play_second 顯示第 frame_idx 個畫面



TimeWarpHardConstraint_t(drop_ball_catch_frame_idx,
desired_catch_frame_idx*time_step)

```
// def of hard constraint (frame_idx,play_second): play frame frame_idx at time play_second
const kinematics::TimeWarpHardConstraintColl_t time_warp_hard_constraint_coll =
{
    kinematics::TimeWarpHardConstraint_t(int32_t{0}, double{0.0}),
    // catch ball motion frame:drop_ball_catch_frame_idx is played at time:drop_ball_catch_frame_idx*time_step
    // time warpped -> catch ball at frame:desired_catch_frame_idx i.e. time:desired_catch_frame_idx*time_step
    // hard constraint = (drop_ball_catch_frame_idx,desired_catch_frame_idx*time_step)
    kinematics::TimeWarpHardConstraint_t(
        param_->value<double>("time_warp.drop_ball_catch_frame_idx"),
        param_->value<int32_t>("time_warp.desired_catch_frame_idx")
        * acclaim::Motion::time_step()
        ),
    kinematics::TimeWarpHardConstraint_t(
        (fk_solver_coll_->at(0)->motion()->frame_num() - 1),
        boost::numeric_cast<double>((fk_solver_coll_->at(0)->motion()->frame_num() - 1))
        * acclaim::Motion::time_step()
        ),
};
```

```cpp
math::SpatialTemporalVector6d_t TimeWarper::ComputeWarpedMotion(
        const TimeWarpHardConstraintColl_t &hard_constraint_coll
        )
{
    *hard_constraint_coll_ = hard_constraint_coll;

    // warp[frame_idx] = new play_second for oringinal motion capture frame
    double *warp_playsec = new double[original_motion_sequence_->temporal_size()];

    for (int i =1; i < hard_constraint_coll_->size(); ++i) {
        int total_frame = hard_constraint_coll_->at(i).frame_idx - hard_constraint_coll_->at(i - 1).frame_idx;
        double total_time = hard_constraint_coll_->at(i).play_second - hard_constraint_coll_->at(i - 1).play_second;
        // uniformly sampled
        double time_step = total_time / total_frame;
        for (int t = 0; t <= total_frame; ++t) {
            int idx = hard_constraint_coll_->at(i - 1).frame_idx + t;
            warp_playsec[idx] = hard_constraint_coll_->at(i - 1).play_second + t*time_step;
        }
    }
```

```cpp
    math::SpatialTemporalVector6d_t new_motion_sequence(original_motion_sequence_->spatial_size(), original_motion_sequence_->temporal_size());

    int scanned = 0;
    for (int t = 0; t < original_motion_sequence_->temporal_size(); ++t) {
        double now = t*time_step_;

        // warp_playsec[frame1] <= now < warp_playsec[frame2]
        for (; scanned < original_motion_sequence_->temporal_size() && now > warp_playsec[scanned + 1]; ++scanned) {}
        int frame1 = (scanned < original_motion_sequence_->temporal_size()) ? scanned : original_motion_sequence_->temporal_size() - 1;
        int frame2 = (frame1 < original_motion_sequence_->temporal_size() - 1) ? frame1 + 1 : original_motion_sequence_->temporal_size() - 1;

        double ratio = 1;
        if (warp_playsec[frame2] - warp_playsec[frame1] > 0) {
            ratio = (now - warp_playsec[frame1]) / (warp_playsec[frame2] - warp_playsec[frame1]);
        }

        for (int s = 0; s < original_motion_sequence_->spatial_size(); ++s) {
            math::Vector6d_t amc1 = original_motion_sequence_->element(s, frame1);
            math::Vector6d_t amc2 = original_motion_sequence_->element(s, frame2);
            // translation linear interpolation
            math::Vector3d_t t1(amc1[3], amc1[4], amc1[5]);
            math::Vector3d_t t2(amc2[3], amc2[4], amc2[5]);
            math::Vector3d_t mixt = math::Lerp(t1, t2, ratio);
            // rotation spherical linear interpolation
            math::Vector3d_t r1(amc1[0], amc1[1], amc1[2]);
            math::Vector3d_t r2(amc2[0], amc2[1], amc2[2]);
            // euler angle -> quaternion
            math::Quaternion_t q1 = math::ComputeQuaternionXyz(math::ToRadian(r1[0]), math::ToRadian(r1[1]), math::ToRadian(r1[2]));
            math::Quaternion_t q2 = math::ComputeQuaternionXyz(math::ToRadian(r2[0]), math::ToRadian(r2[1]), math::ToRadian(r2[2]));
            // slerp(q1,q2,r) = [sin((1-r)*thata)/sin(thata)]*q1 + [sin(r*theta)/sin(thata)]*q2
            math::Quaternion_t mixq = math::Slerp(q1, q2, ratio);
            // quaternion -> rotation matrix -> euler angle
            math::Vector3d_t mixr = math::ToDegree(math::ComputeEulerAngleXyz(math::ComputeRotMat(mixq)));
            math::Vector6d_t mixamc(mixr, mixt);

            new_motion_sequence.set_element(s, t, mixamc);
        }
    }

    delete warp_playsec;

    return new_motion_sequence;
}
```

♦ Result and Discussion

● 在 forward kinematics 的部分：ASF 的資料，全部都是參考 global coordinate；AMC 的資料，除了 root 是參考 global coordinate，其餘則是參考各自的 local coordinate。

● 在 time warp 的部分：Tx,Ty,Tz 使用 linear interpolation = Lerp，Rx,Ry,Rz 使用 spherical linear interpolation = Slerp，必須先把 Eular angle 轉換成 Quaternion 才能進行運算。使用 Quaternion 可避免 gimbal lock，不過在資料不會造成該問題的時候，使用 Lerp 與 Slerp 計算 Rx,Ry,Rz 所產出的動畫似乎差異不大。