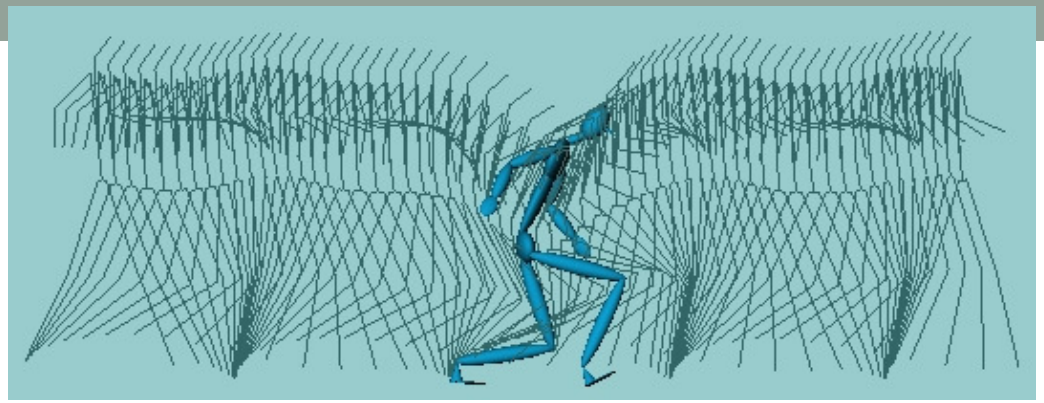
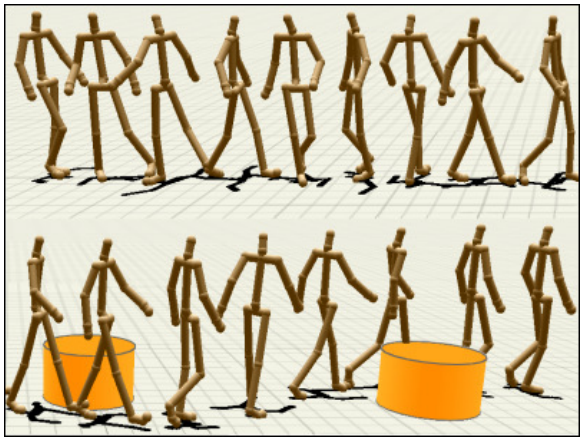


ANIMATION DE PERSONNAGE BASÉE SQUELETTE

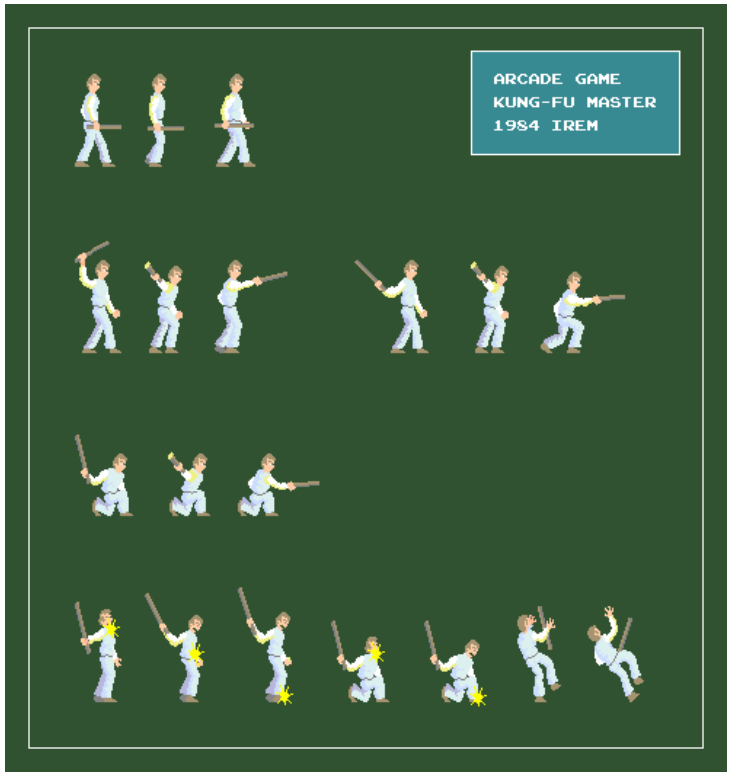
Alexandre Meyer

Master Informatique

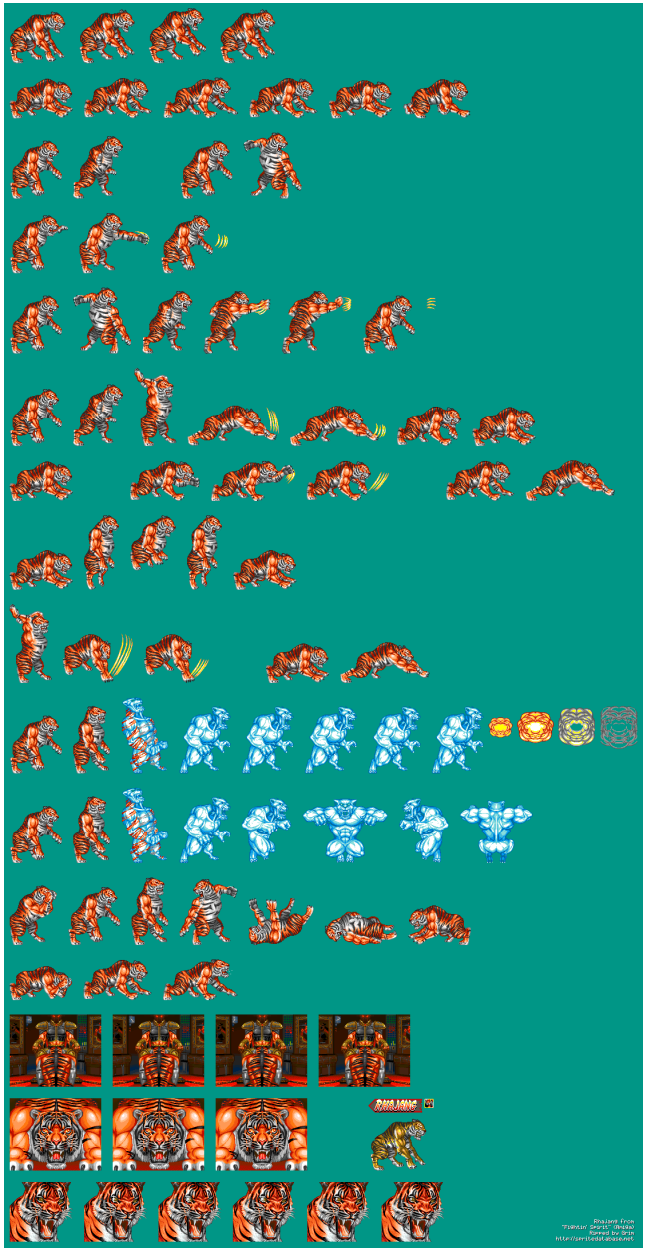


Historiquement en 2D

Série de sprites
avec les positions clés



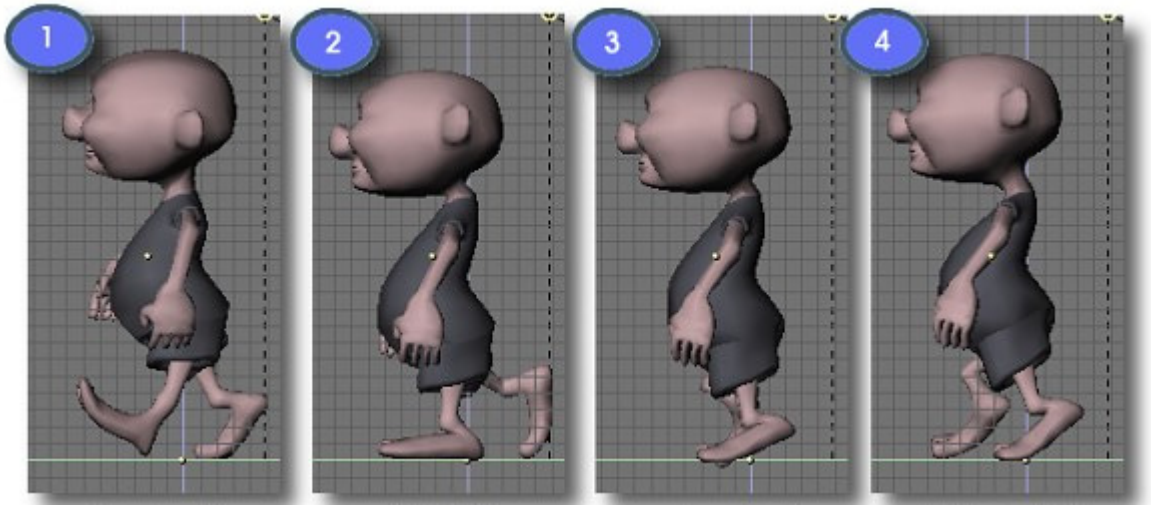
+ VIDEO



Idée 1 : Animation personnage 3D

- 1ère manière d'animer un personnage
  - Série de maillage correspondant à des positions clés de l'animation

Affiche : mesh1, mesh2, mesh3, mesh4, mesh1, mesh2, etc.

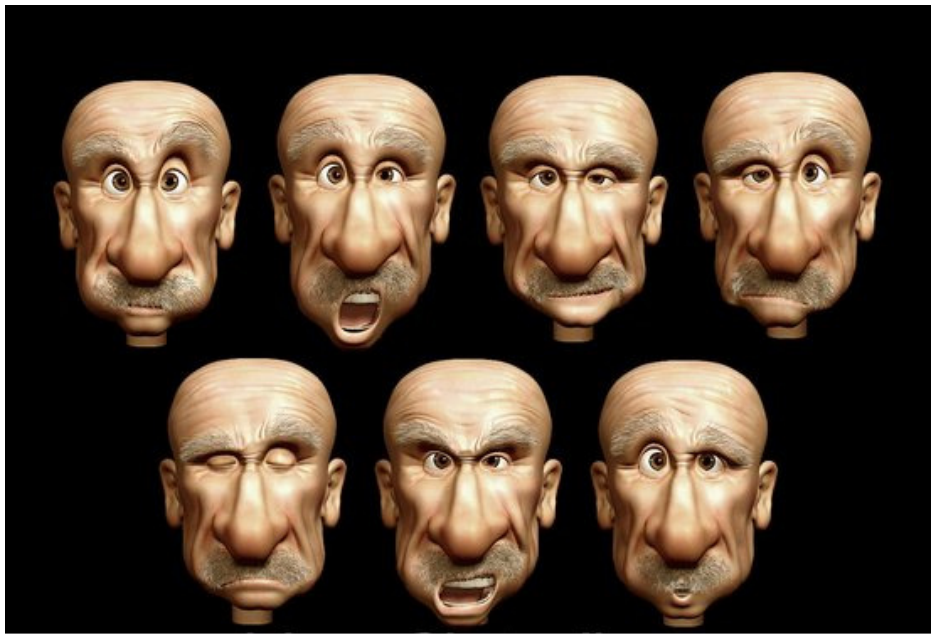


Idée 1 : Animation personnage 3D

- 1ère manière d'animer un personnage
  - Positions clés de l'animation
  - Interpolation de maillage (BlendShape)
  - Beaucoup utilisé pour les visages
- Inconvénients : création des données et stockage

+ VIDEO

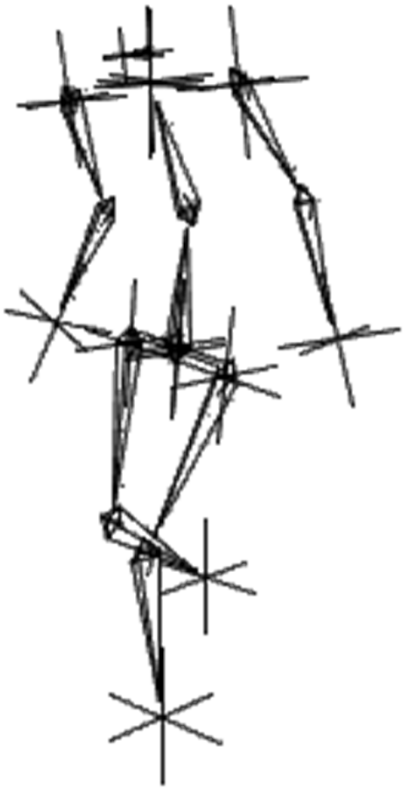
3% mesh1
+ 40% mesh2
+ 17% mesh3
+ …

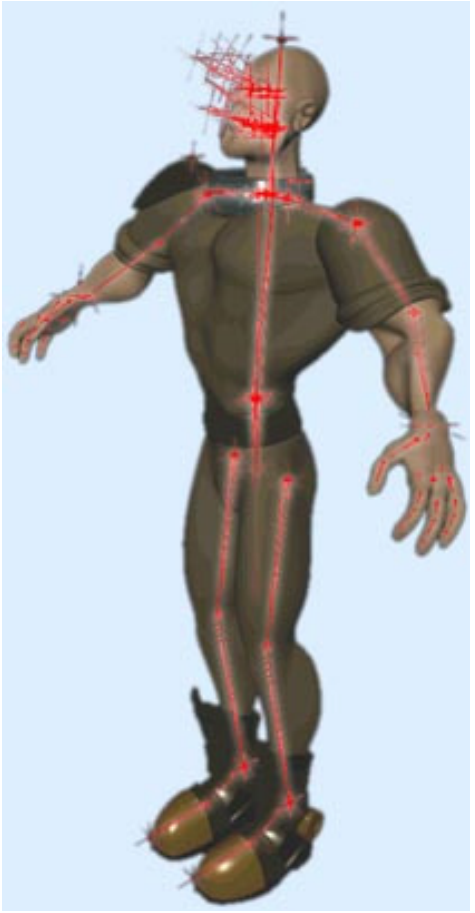


Idée 2 : Animation personnage 3D

2e manière d'animer un personnage

- Personnage 3D = 1 squelette + 1 maillage(avec texture)

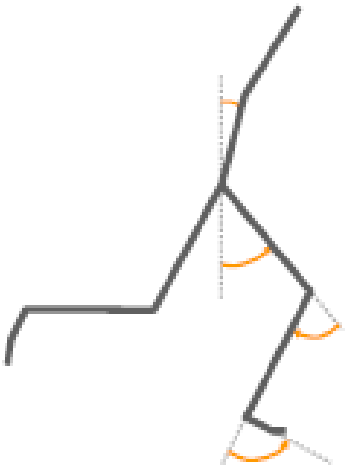


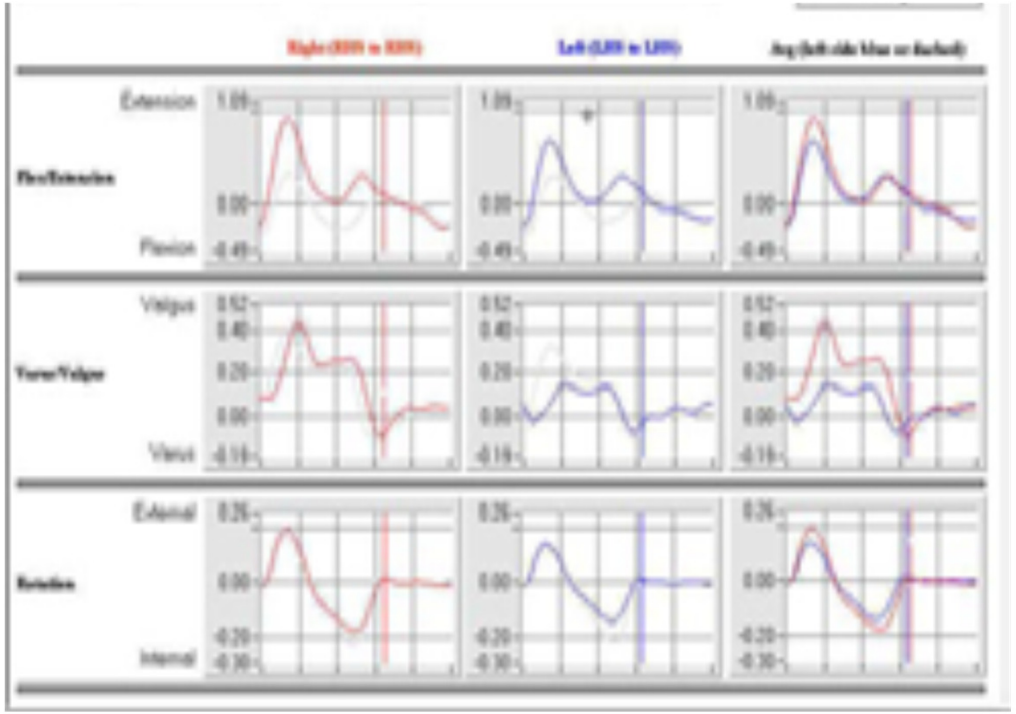


Idée 2 : Animation personnage 3D

Interpolation des positions clefs du squelette

+Déformation du maillage (skinning)





Idée 2 : Vocabulaire

- Manipulation directe (**cinématique directe**)
  - Angles → squelette en 3D
- ***Skinning*** pour déformer le maillage
- **Cinématique inverse**
  - Position extrémité → calcul angles

Idée 2 : Animation personnage 3D

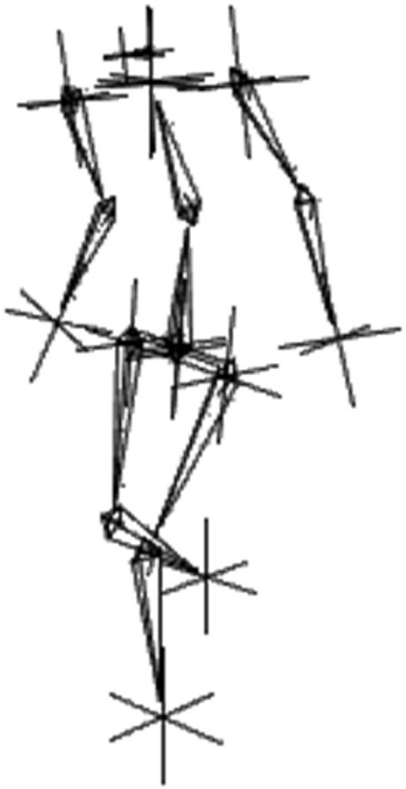
Intérêts de l'animation basé squelette

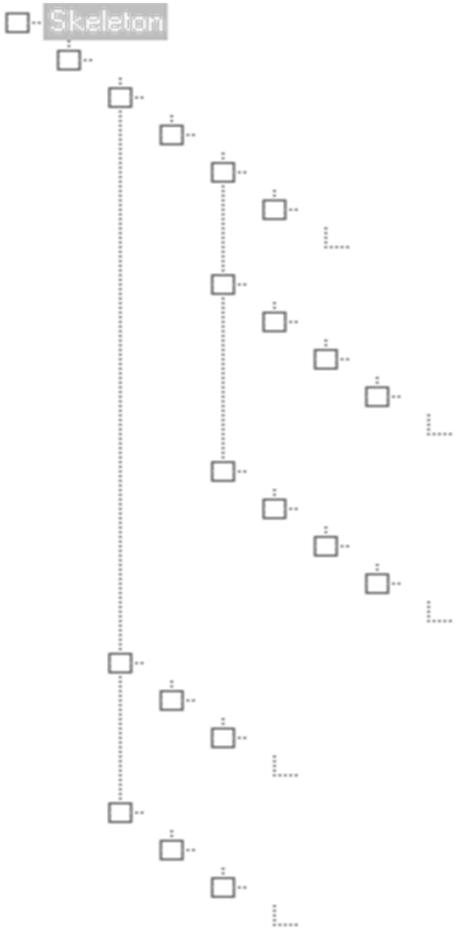
- Squelette
  - Structure hiérarchique sur l'objet

    Epaule tourne ➔ tout le bras tourne
  - Plus facile de capturer le squelette d'un acteur que sa surface
    →*Motion Capture (MoCap)*
- Déformation du maillage
  - Peu couteux en stockage (1 maillage+qq infos)
  - Calcul de déformation suffisamment simple pour être temps réel (et possible sur GPU)

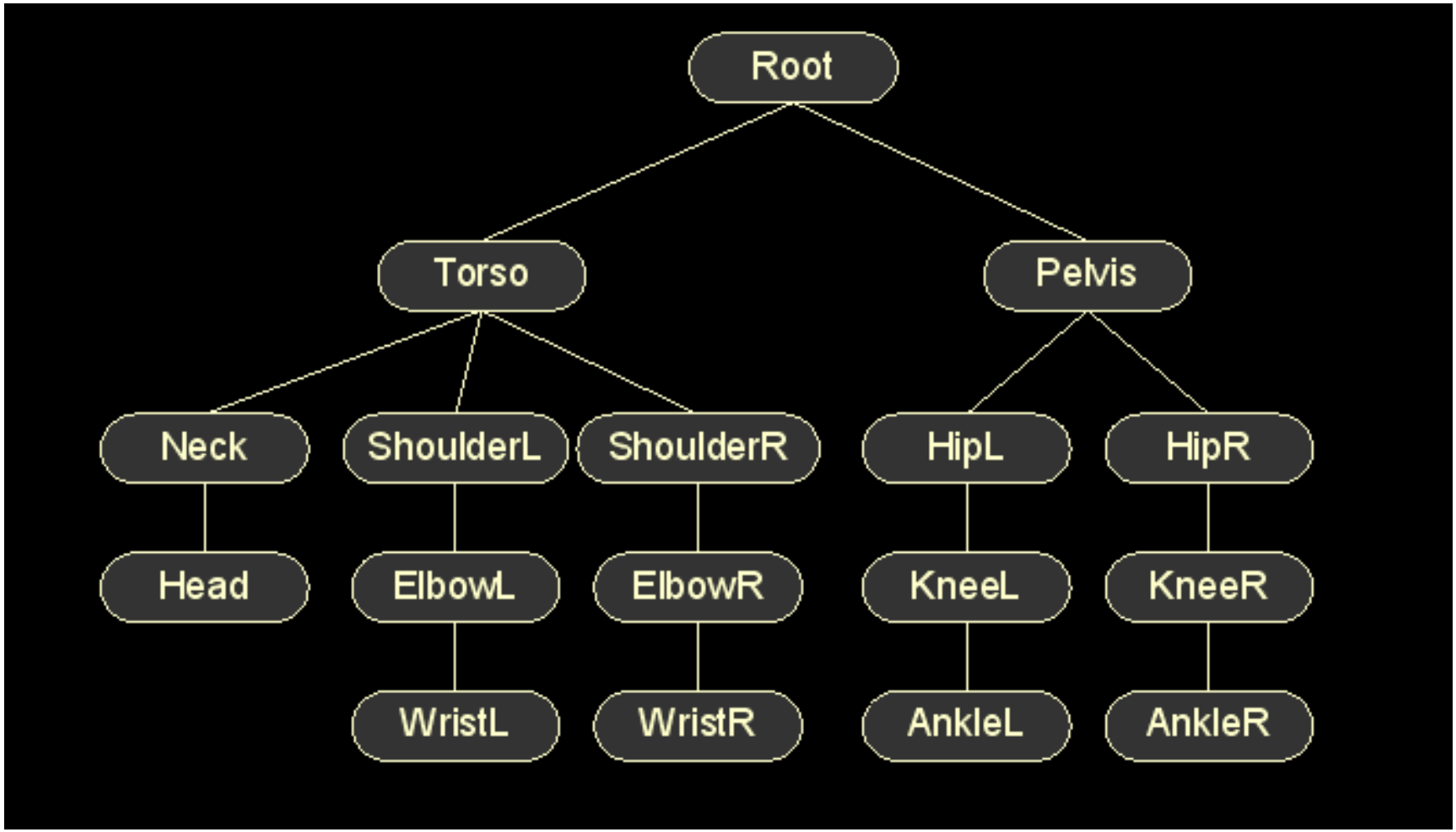
Squelette

- Pas de géométrie, que des « os »
- Notion de hiérarchie (arbre)



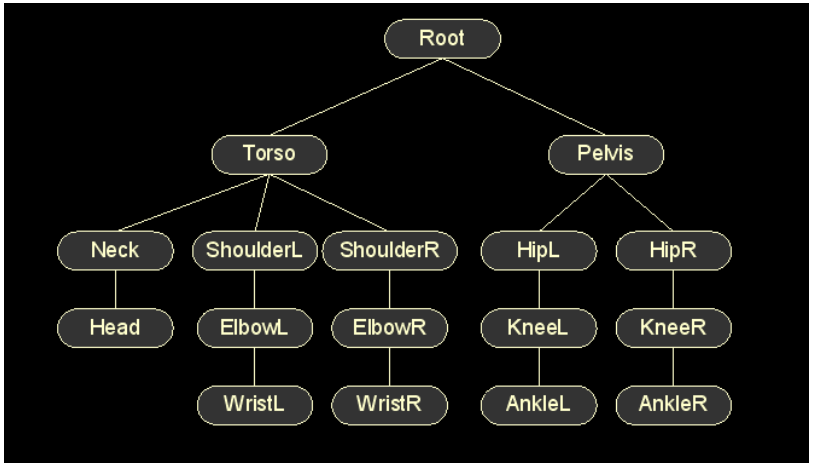


Squelette = arbre



Squelette = arbre

- 1 nœud de l'arbre = 1 ***articulation, os, joint ou bone***
  - une matrice relative au père : local2father (l2f)
  - Matrice = rotation + translation
    - Rotation = orientation de l'os
    - Translation → position au repos
- La matrice est contrainte selon le type de joint
  - Ex. coude simple = 1-DOF (degree of freedom)
  - Os de longueur fixe → translation constante durant anim



Intérêts des arbres

- Structure hiérarchique sur l'objet
  - L'épaule tourne ➔ tout le bras tourne
- Similaire à l'idée de graphe de scène
- Boites englobantes
  - Construites hiérarchiquement
  - Collision, contact
  - Affichage/LOD
- Édition interactive du modèle
  - Animer un squelette est plus simple que d'animer un maillage

Affichage récursif du squelette

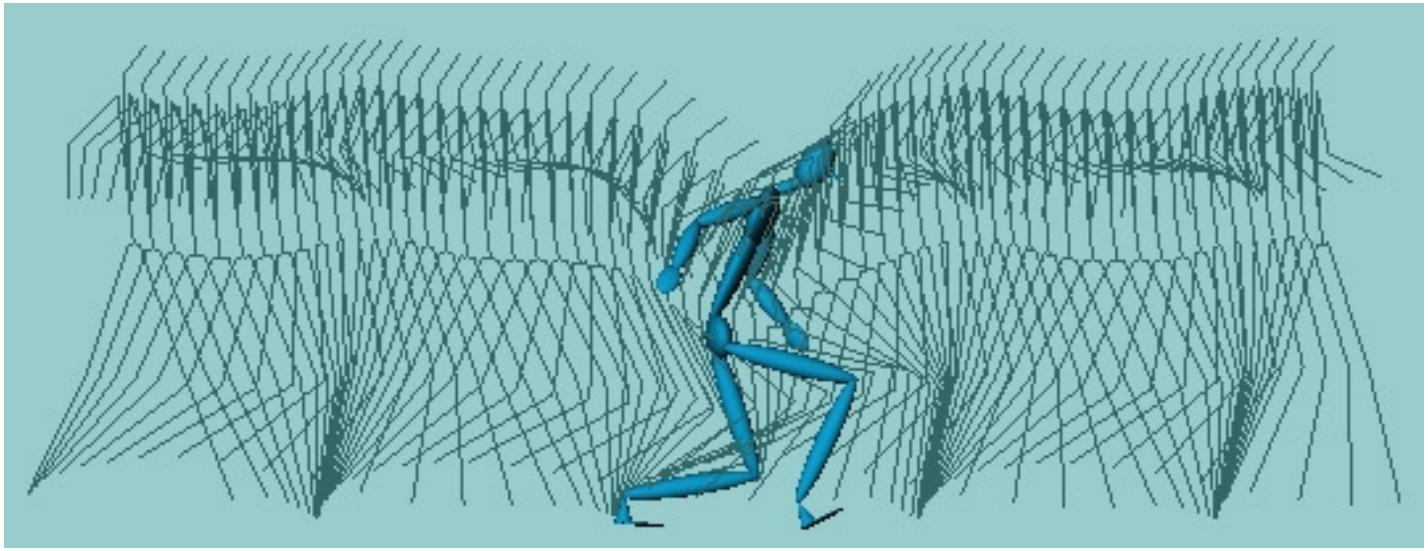
Affichage du squelette = parcours de l'arbre (OpenGL 1.0)

```
void draw(node)
{
  glPushMatrix();
  glTranslate(…,…,…);
  glRotate(…,  …,…,…);
  drawGeometry(node);
  for (i=0; i<numChildren; i++)
    draw(children[i]);
  glPopMatrix();
}
```

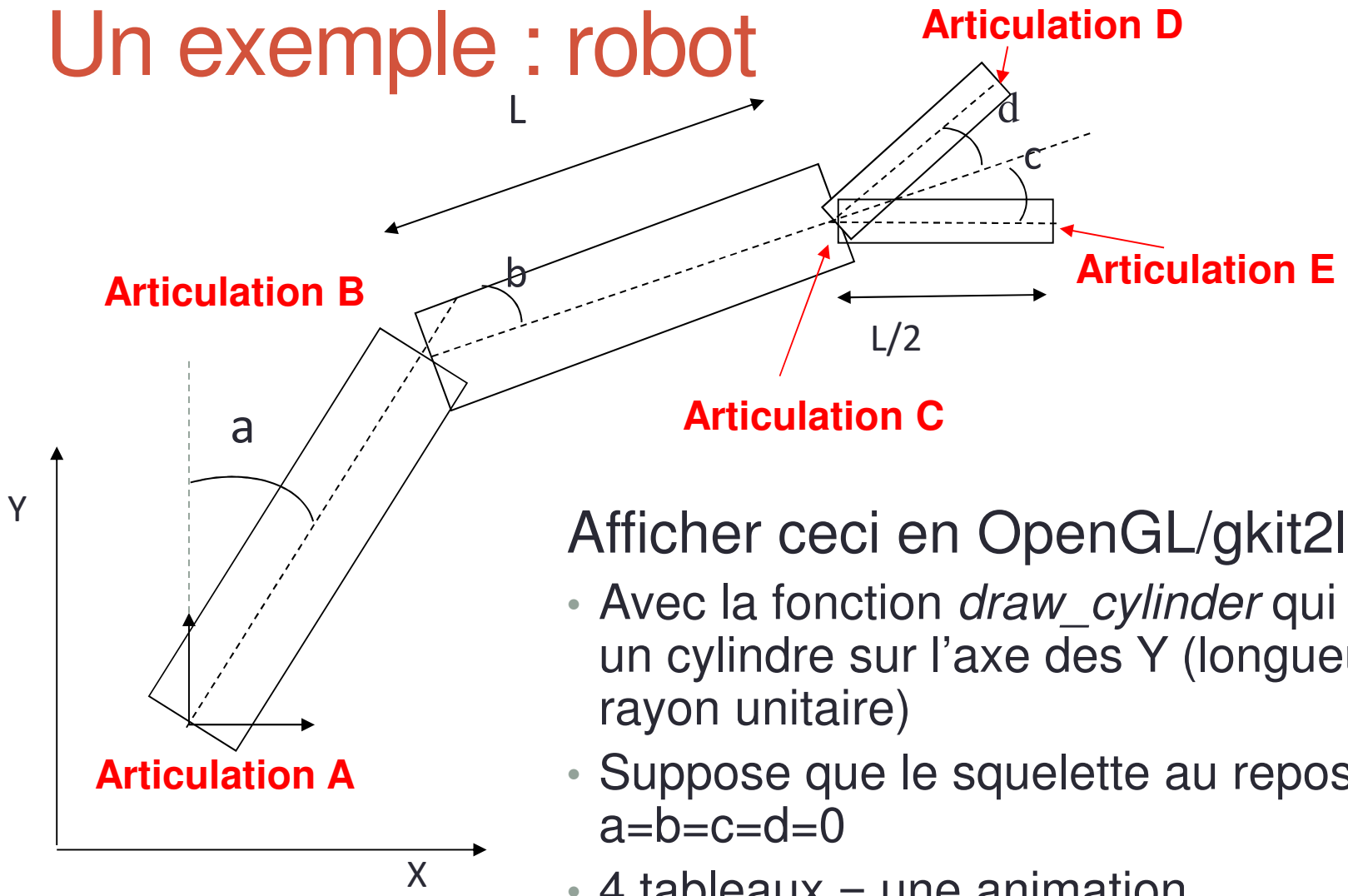
Animation par positions clés du squelette

- Animation par position clés (key framing)
  - Position initiale (au repos) = un squelette à J joints
  - Une pose = J rotations (avec 1 rotation=1 angle, 2 angles ou 3 angles)
  - Une animation = n poses

Pose au repos



Un exemple : robot

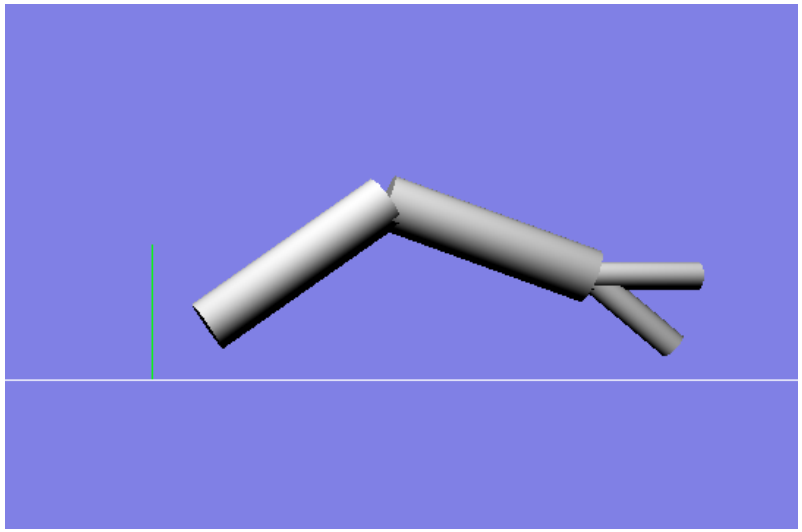


Afficher ceci en OpenGL/gkit2light

- Avec la fonction *draw_cylinder* qui affiche un cylindre sur l'axe des Y (longueur et rayon unitaire)
- Suppose que le squelette au repos est a=b=c=d=0
- 4 tableaux = une animation
  - float A[100], B[100], C[100], D[100];

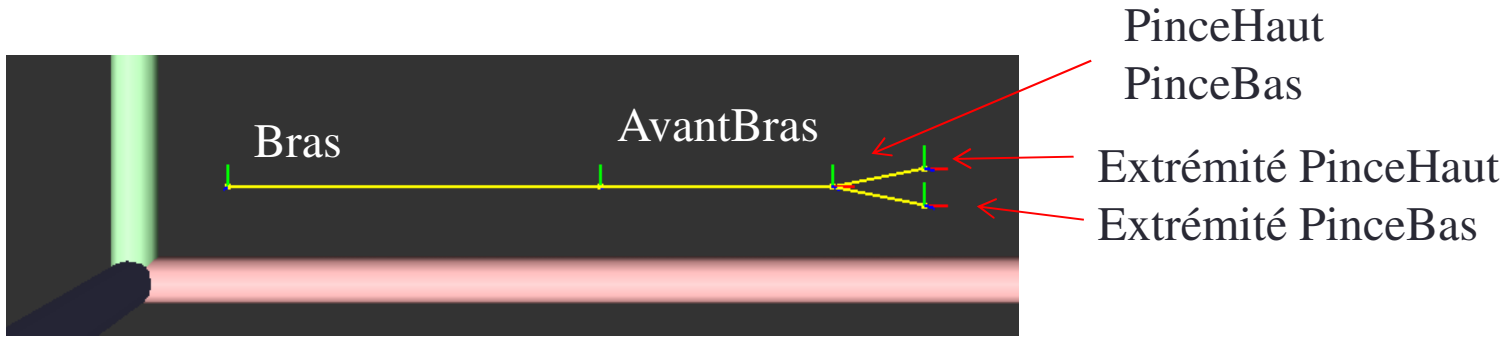
Un exemple : robot simple

```
const int N = 10;
float A[N] = { -10, -15, -20, -35, -40, -45, -50, -55, -70, -75};
float B[N] = { -10, -15, -20, -35, -40, -45, -50, -55, -70, -75 };
static int t = 0;
t = (t + 1) % N;
float a = A[t]; // -45;
const float b = B[t];  -20;
const float c = 20;
const float d = -20;

Transform a2w = Translation(2, 2, 0) * RotationZ(a);
draw_cylinder(a2w * Scale(1,8,1) );

Transform b2a = Translation(0, 8, 0) * RotationZ(b);
Transform b2w = a2w * b2a;
draw_cylinder(b2w * Scale(1, 8, 1));

Transform c2b = Translation(0, 8, 0) * RotationZ(c);
Transform c2w = b2w * c2b;
draw_cylinder(c2w * Scale(0.5, 4, 0.5));

Transform d2b = Translation(0, 8, 0) * RotationZ(d);
Transform d2w = b2w * d2b;
draw_cylinder(d2w * Scale(0.5, 4, 0.5));
```



Description plus générale : BVH

Position au repos



- Bras

Origine du bras par rapport au monde=(20,20,0)

- AvantBras

Origine de l'AvantBras par rapport au Bras=(80,0,0)

- **PinceHaut**

Origine de PinceHaut=(50,0,0) par rapport à AvantBras

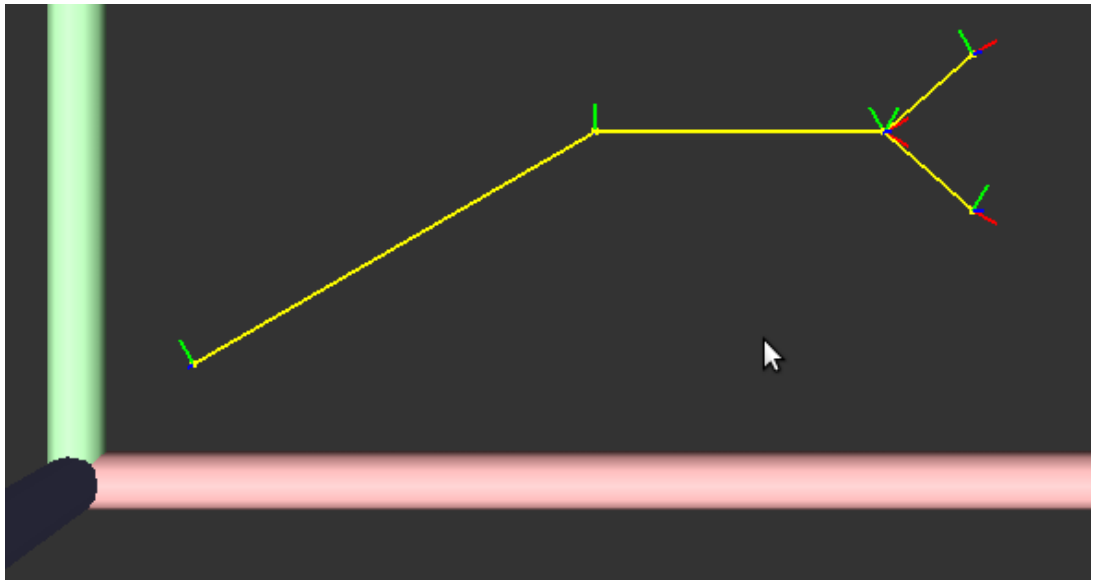
Extrémité de PinceHaut par rapport origine de PinceHaut=(20,2,0)

- **PinceBas**

Origine de PinceBas=(50,0,0) par rapport à AvantBras

Extrémité de PinceBas par rapport origine de PinceBas=(20,-2,0)

Description plus générale : robot



- Animation : 4 paramètres
  - Angle Bras (par rapport au repos)
  - Angle AvantBras
  - Angle PinceHaut
  - Angle PinceBas

Description plus générale : BVH=format de fichier

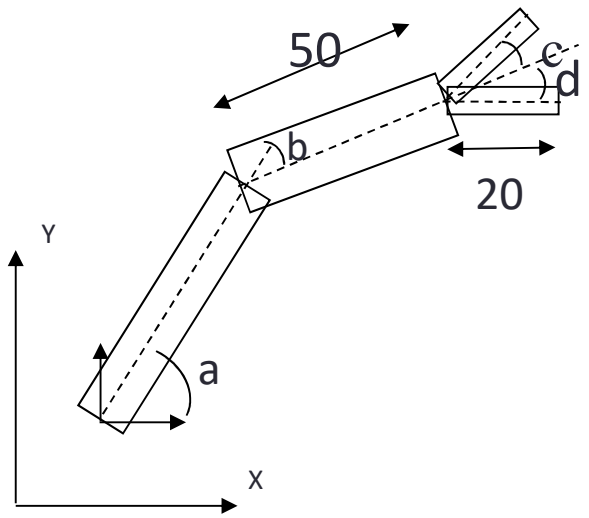
```
ROOT Bras
  OFFSET          20.00                20.00                0.00
  CHANNELS 1 Zrotation
  JOINT AvantBras
          OFFSET   80.00   0          0.00
          CHANNELS 1 Zrotation
          JOINT PinceHaut
                  OFFSET   50.00   0          0.00
                  CHANNELS 1 Zrotation
                  End Site  OFFSET           20.00     4          0.00
          JOINT PinceBas
                  OFFSET   50.00   0          0.00
                  CHANNELS 1 Zrotation
                  End Site OFFSET            20.00     -4         0.00
```

Description du squelette au repos

```
HIERARCHY
ROOT Bras
{
    OFFSET      20.00                           20.00                           0.00
    CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
    JOINT AvantBras
    {
                OFFSET          80.00           0               0.00
                CHANNELS 3 Zrotation Xrotation Yrotation

                JOINT PinceHaut
                {
                            OFFSET          50.00           0               0.00
                            CHANNELS 3 Zrotation Xrotation Yrotation
                            End Site
                            {
                                        OFFSET          20.00           4               0.00
                            }
                }

                JOINT PinceBas
                {
                            OFFSET          50.00           0               0.00
                            CHANNELS 3 Zrotation Xrotation Yrotation
                            End Site
                            {
                                        OFFSET          20.00           -4              0.00
                            }
                }
    }
}
MOTION
Frames:    4
Frame Time: 0.33333
0 0 0 10 0 0     -10 0 0     10 0 0     -10 0 0
0 0 0 20 0 0     -20 0 0     20 0 0     -20 0 0
0 0 0 30 0 0     -30 0 0     30 0 0     -30 0 0
```

```
HIERARCHY
ROOT Bras
{
    OFFSET        20.00   20.00    0.00
    CHANNELS   6   Xposition   Yposition   Zposition   Zrotation   Xrotation   Yrotation
    JOINT AvantBras
    {
                OFFSET          80.00    0    0.00
                CHANNELS   3    Zrotation   Xrotation   Yrotation

                JOINT PinceHaut
                {
                            OFFSET          50.00   0   0.00
                            CHANNELS   3    Zrotation   Xrotation   Yrotation
                            End Site
                            {
                                        OFFSET          20.00   4   0.00
                            }
                }

                JOINT PinceBas
                {
                            OFFSET          50.00   0   0.00
                            CHANNELS   3    Zrotation   Xrotation   Yrotation
                            End Site
                            {
                                        OFFSET          20.00   -4   0.00
                            }
                }
    }
}
MOTION
Frames:    4
Frame Time: 0.33333
0 0 0 10 0 0      -10 0 0      10 0 0      -10 0 0
0 0 0 20 0 0      -20 0 0      20 0 0      -20 0 0
0 0 0 30 0 0      -30 0 0      30 0 0      -30 0 0
```

a b c d



+ autre exemple de .bvh

BVH et matrice

```
Joint PinceHaut {
  OFFSET      50.00        10.00        2.00
  CHANNELS 1 Zrotation
```

Signifie en GL

glTranslationf(50 ,10 , 2)

glRotatef(a, 0, 0, 1) ; // a=angle_des_données_MOTION)

Ou

$$\begin{pmatrix} \cos(a) & -\sin(a) & 0 & 50 \\ \sin(a) & \cos(a) & 0 & 10 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix} = Matrice_{Bone_{i-1}<-Bone_i} = Matrice_{pere<-local}$$

Affichage récursive du squelette

Affichage OpenGL 1.0 du squelette issue du BVH précédent

```
void draw(BVHJoint joint)
{
  glPushMatrix();
  Draw_Bone();// draw line from (0,0,0) to (OffsetX,OffsetY,OffsetZ)
  glTranslate(OffsetX, OffsetY, OffsetZ);
  glRotate( a,  0,0,1);
  for (i=0; i<joint.numChildren; i++)
    draw( joint.children[i]);
  glPopMatrix();
}
```

Affichage récursive du squelette

Question 1 du TP : affichage gkit2light du squelette

```
void bvhDraw(BVHJoint joint, const Transform& f2w)
{
  Transform l2f = … //Translation Offset + Rotation décrites par
    les channels

  Transform l2w = f2w * l2f;

  // Dessine Articulation = sphere + ligne du nœud courant
  // vers le père donc de l'origine de l2w à l'origine de f2w
  draw_sphere( …
  draw_cylinder( …

  for (i=0; i<joint.numChildren; i++)
    bvhDraw( joint.children[i], l2w);
}
```

Affichage non récursif (Question 2 du TP)

Nécessaire pour le skinning ou la comparaison de pose dans le graphe d'animation ou pour le calcul sur GPU

```
struct SkeletonJoint
{
    int m_fatherId;                 // Le numéro du père dans le tableau de CAJoint de CASkeleton
    Mat4f   m_local2world;          // La matrice passant du repère de l'articulation vers le monde
};

class Skeleton {
public:
    //! Créer un squelette ayant la même structure que définit dans le BVH
    void init(const BVH& );
    //! Positionne ce squelette dans la position n du BVH
    void setPose(const BVH& bvh, const int frameNumber);
    //! Calcule la distance entre deux postures. Precond: les deux squelettes doivent être identique
    float distance(const Skeleton& skel) const;
protected:
    //! L'ensemble des articulations. Remarque :hiérarchie (arbre)est remplacé l'information "fatherID"
    vector<SkeletonJoint> m_joint;
};
```

Affichage non récursif (Question 2 du TP)

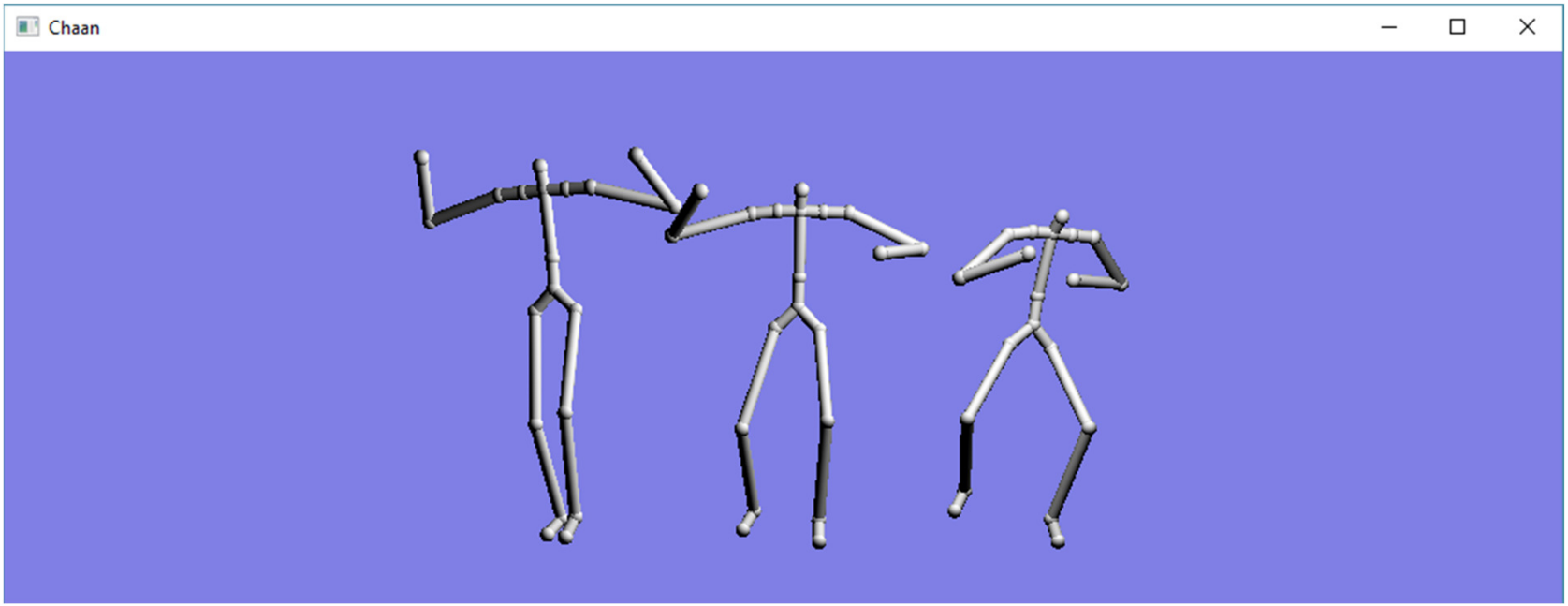
Indice	0	1	2	3	…
Indice du père	-1	0	1	1	
Monde ← Local	$M_{monde \leftarrow 0}$	$M_{monde \leftarrow 1} = M_{monde \leftarrow 0} \times M_{0 \leftarrow 1}$	$M_{monde \leftarrow 2}$	$M_{monde \leftarrow 3} = M_{monde \leftarrow 1} \times M_{1 \leftarrow 3}$	

Pour obtenir la matrice 4x4 (Transform) m_local2world; // monde<-local

- On fait
  - $M_{monde<-local} = M_{monde<-pere} \times M_{pere<-local}$
  - Pour une articulation (*joint)* i
    - Transform l2f = … ; //On construit la matrice local2father avec le transparent 21
    - int pere_i = m_joints[i].fatherId;
    - m_joints[i].m_local2world = m_joints[pere_i].m_local2world * l2f;

Interpolation entre 2 poses

- Pour passer d'une animation à une autre
- Pour gérer le temps de manière exacte entre deux poses d'une même animation
- Voir TP
  - Au milieu = interpolation entre pose de gauche et pose de droite



Interpolation

- Interpolation entre 2 poses d'animations
  - ne pas interpoler les positions 3D (sinon longueur des membres non garantie)
  - Interpolation des angles, Possible mais attention
    - angle1 = -10
    - angle2 = 10 ==> angle_milieu = (-10+10)/2 = 0
    - Mais
      - angle1=angle1' = 360-10 = 350 (identique à l'angle 1)
      - angle2 = 10 ==> angle_milieu = (350+10)/2 = 180 != 0 (!!!!!!)
  - Interpolation des rotations en passant par des quaternions
    - mieux

Matrice vs Quaternion/Translation

- Dans notre cas, les matrices comportent une Rotation et une Translation (pas de changement d'échelle ou autre)
  - Transform M_A

$$M_A = \begin{pmatrix} R_A & T_A \\ 0 & 1 \end{pmatrix} \quad \text{avec} \quad R_A = \begin{pmatrix} - & - & - \\ - & - & - \\ - & - & - \end{pmatrix} = rotation$$

$$\text{et} \quad T_A = \begin{pmatrix} Tx \\ Ty \\ Tz \end{pmatrix} = vecteur\_translation$$

  - On peut représenter ceci avec un quaternion + vecteur
    - Quaternion Q_A; // cf. cours Quaternion
    - Vec3f T_A; // http://fr.wikipedia.org/wiki/Quaternions_et_rotation_dans_l'espace

Matrice VS Quaternion/Translation

- Combiner des transformations
  - Mat4f M_a;
  - Mat4f M_b;
  - Mat4f M_ab = M_a * M_b;

$$M_a = \begin{pmatrix} R_a & T_a \\ 0 & 1 \end{pmatrix}$$

$$M_b = \begin{pmatrix} R_b & T_b \\ 0 & 1 \end{pmatrix}$$

$$M_a \times M_b = \begin{pmatrix} R_a & T_a \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} R_b & T_b \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_a \times R_b & R_a \times T_b + T_a \\ 0 & 1 \end{pmatrix}$$

- Transformation (=Quaternion + vecteur) A et B
  - Quaternion Q_a,; Vec3f T_a, // Transfo A
  - Quaternion Q_b; Vec3f T_b; // Transfo B
  - Composition A o B

➔ Quaternion Q_ab = Q_a * Q_b;

Vec3f T_ab = Q_a * T_b + T_a;