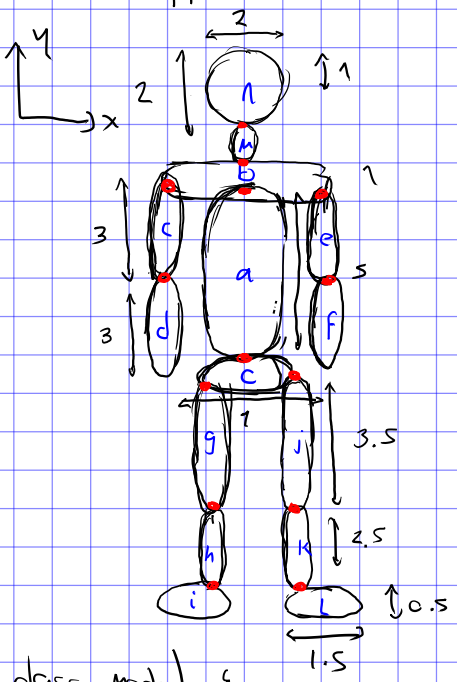
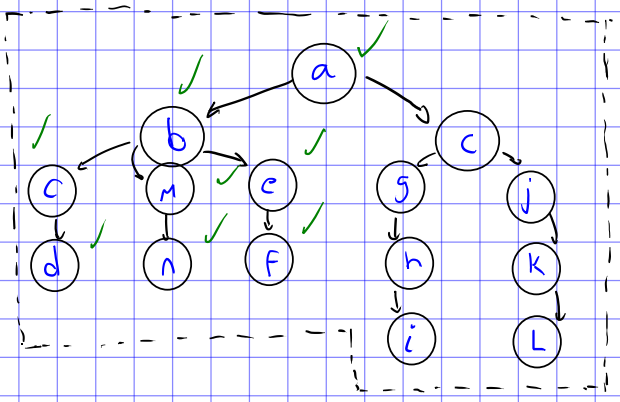
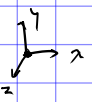
Ze puppet :



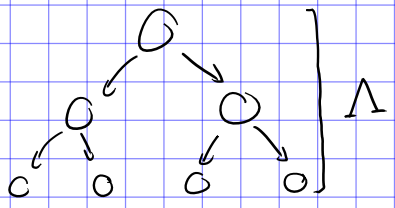
Puppet tree : Λ



$a \propto \vec{0}$



class model {



};

ellipsoid {

vec3 origin
vec3 scale
vec3 position
vec3 rotation (con respecto al origen)

vec3 color

}

genCircleVertex → S^2

DrawModel(M, View, Projection)

VertexArray ← VertexBuffer ← S^2

traverse(M.tree)

Model $\in M_{4x4}(\mathbb{R})$

$M = mat4(rotate) \times mat4(origin) \times mat4(scale)$

$M = mat4(position) \times M$

Shader ← M, View, Projection, vec4(color, 1f)

render(va, vb, shader)

```
Node {
    string name
    ellipsoid part
    vector<node> children
    bool endeffector
};
```

```
Model {
    Node head
};
```

```
Model puppet

puppet.translate("torso", someMove)

    trans = mat4(vec3(someMove))
    now = this->head
    while( now.name = "torso" or now = endeffector)
        now.part pushdown(trans)
```

Repite esto mismo para rotate...

- Añadir bandera para restringir movimiento (en node).
  - Translate:
  - Rotate:

Ahora sob tengo que averiguar como hacer iluminacion y camara

:)

Done owo

```
MatrixStack {
    mat4 stack = I
    stack operation_stack
    push(A) {
        operation_stack.push(A)
        stack *= A
    }
    pop() {
        A = operation_stack.top();
        stack *= A^-1
        operation_stack.pop();
    }
}
```

Model push/pop del libro (rosa, morado)

UNU

esto es una mala idea !

Sea $n := $ "numero de primitivas del modelo" pero calculo inversa...

Complejidad de tiempo $O(n \log n)$ ← too bad!

Complejidad de memoria $O(2n^2 \log n)$

Mejor...

operation-stack → $\prod_{i=0}^{n} A_i$ ← top

$A_2 A_1 A_0$	2
$A_1 A_0$	1
A_0	0

⇒ $ops.push(B) = B * S_n$

$ops.pop() = B^{-1} S_{n+1}$

$= S_n$

Ejem:

$$A_3 A_2 A_1 A_0 = S_n$$

$$B\ A_3 A_2 A_1 A_0 = S_{n+1}$$

$$\underbrace{B^{-1}B}_{I}\ A_3 A_2 A_1 A_0 = S_n$$

```
operation-stack {
    vector os
    push(A) {
        mat N = A * os.top()
        os.push(N)
    }
    pop() {
        os.pop()
    }
    top() {
        os.top
    }
}
```

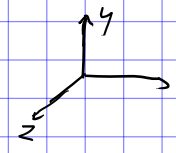
No hay inversa!

en realidad solo es un wrapper de std::stack

Casi lo olvido...

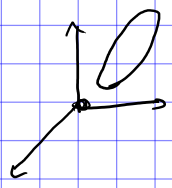
Un objeto tiene

Atributos de su frame



← Position (origin position)
orientation
scale

y Atributos respecto a su frame...



position
scale
rotation

todos los atributos de frame son dependientes del padre

Entonces, sea $R(\theta, v)$, $S(v)$, $P(v)$ los generadores de matrices afines de rotación, escala y traslación, la matriz de transformación de frame es,

$$A = P(v_f)\, R(\theta, v_F)\, S(v)$$

Luego, sea $\bar{o}^T$ la matriz de Frame del padre, entonces la matriz del frame del objeto

$$\bar{a}^T = \bar{c}^T A \ldots$$

Después se calcula B idéntica a A para transformaciones locales y

$$\text{mat4 Model} = \bar{a}^T B$$

Al siguiente nivel se pasa $\bar{a}^T$ y repite...

Pero ahora esto:

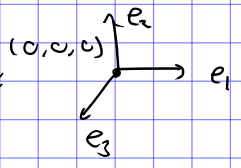
```
push(A){
  mat N = A * OS.top()
  OS.push(N)
}
```

Pasa a esto

```
push(A){
  mat N = OS.top * A
  OS.push(N)
}
```

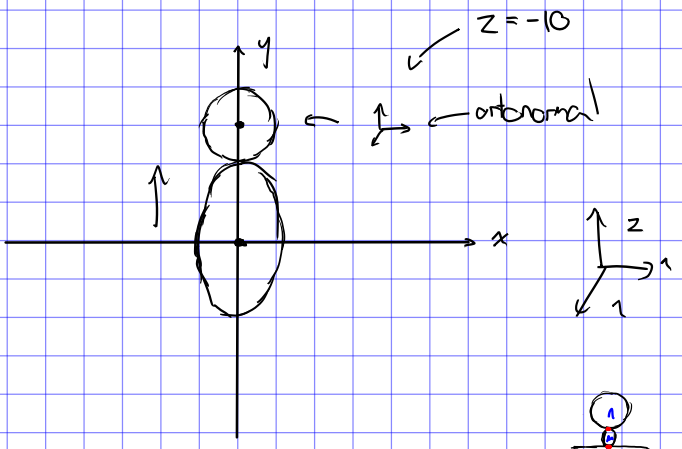
Ya que las matrices frame son del tipo

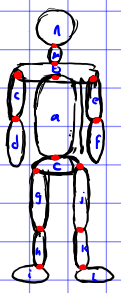
$$\bar{c}^T = \bar{w}^T O c$$

Además $\bar{C}^T = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \approx$ 

sud. tengo un mejor plan

Test





Puppet Render test:

```
Drawing "torso"
Drawing "hips"
Drawing "leftUpperLeg"
Drawing "leftLowerLeg"
Drawing "leftFoot"
Drawing "rightUpperLeg"
Drawing "rightLowerLeg"
Drawing "rightFoot"
Drawing "shoulders"
Drawing "neck"
Drawing "head"
Drawing "leftArm"
Drawing "leftForearm"
Drawing "rightArm"
Drawing "rightForearm"
```