ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL

Facultad de Ingeniería en Electricidad y Computación

"REALIDAD AUMENTADA PARA LA ENSEÑANZA DE INGENIERÍA"

INFORME DE PROYECTO INTEGRADOR

PREVIO A LA OBTENCIÓN DEL TÍTULO DE:

INGENIERO EN CIENCIAS COMPUTACIONALES

DAVID ALEJANDRO BARRERA GUANO

GUAYAQUIL – ECUADOR

AÑO: 2016

AGRADECIMIENTOS

Mis más sinceros agradecimientos primero a mis padres que velaron día a día por mi educación, apoyándome en todo momento, a mi tutor por su tan acertada guía y a mi profesora de materia integradora por sus sabios consejos y a estas personas tan especiales que han logrado una huella en mi vida.

TRIBUNAL DE EVALUACIÓN

Sixto García Aguilar

PROFESOR EVALUADOR

Cristina Abad Robalino

PROFESOR EVALUADOR

DECLARACIÓN EXPRESA

"La responsabilidad y la autoría del contenido de este Trabajo de Titulación, me corresponde exclusivamente; y doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual"

David Barrera Guano

RESUMEN

En este proyecto se presenta el diseño e implementación de un sistema que realiza animaciones para ser utilizadas en avatares humanoides llamado "StickAnimator". StickAnimator fue realizado para mostrar de manera sencilla el concepto de keyframing (frames principales). Adicionalmente, provee también un método sencillo de animar avatares humanoides en 3D, tan solo dibujando sus poses principales a manera de sketches de bolitas y palitos. Para su implementación se utilizó python, opengl y el sistema de ventanas wxpython.

En este proyecto se utilizó el concepto de proporciones del cuerpo humano para poder inferir la tercera dimensión de una pose, además, se utilizó la interpolación de puntos por medio de curvas para encontrar las poses intermedias, por motivos de coste computacional estas curvas no son perfectas, y aunque el problema no es detectable a simple vista, se espera que en un futuro este problema se solucione.

En futuras versiones de este proyecto se espera que la calidad de la interpolación de frames mejore, además de que pueda abrir más tipos de archivo de 3D que solo el actual. Con respecto a proyectos futuros derivados de este, se espera que este mismo método permita disminuir la información requerida para guardar las animaciones de un avatar, además de reusar estas animaciones en avatares distintos y hasta tal vez se podría usarlo para hacer una especie de Kinect con 1 sola cámara.

ÍNDICE GENERAL

CAPÍTULO 1

1. Antecedentes

1.1 Descripción del problema

La tecnología en general ha ayudado a las personas a realizar tareas complicadas de manera sencilla. Antiguamente, por poner un ejemplo, los personajes de caricaturas se las dibujaban a mano; para realizar una animación se debía dibujar cada frame intermedio también a mano. Ahora con el uso de la tecnología no es necesario dibujar un personaje a mano ni redibujarlo en cada frame; es posible crearlo en un modelador 3D y animarlo desde allí, pero el proceso de animar un avatar ya de por sí representa cierta dificultad. A menudo la animación de un personaje en 3D, sobre todo cuando tiene bastantes articulaciones, representa una demanda de bastante paciencia y técnica; una técnica que varía un poco dependiendo de qué modelador se esté usando, y a la final, aún cuando la animación que tenemos en nuestra cabeza la tenemos clara, el resultado no es tan bueno como lo esperábamos. Es necesario alguna herramienta que logre hacer esa conexión de lo que queremos con lo que logramos crear.

1.2 Solución propuesta: reduciendo la complejidad

La propuesta al problema que se enfrenta es hallar una manera más natural de crear una animación, y que dicha herramienta pueda interpretar esa información por nosotros y animar el objeto en 3D en consecuencia, como lo haría cualquier diseñador. La manera más sencilla de expresar una animación es en términos de stickfigures (o personajes de bolitas y palitos) en 2 dimensiones. Esta herramienta podría interpretar estos dibujos como puntos en 2D, dibujar los frames de relleno o frames intermedios y añadir profundidad en estos stickfigures. Una vez hecho esto, girar los bones del objeto en 3D a animar por cada pose generada.

1.3 Objetivos y alcance

Los objetivos de este proyecto son resolver el problema de la animación de un personaje de manera más sencilla y rápida. Este proyecto además haría la tarea más accesible a personas que no estén familiarizadas con la animación de avatares.

En este proyecto se logró hacer una aplicación que facilita considerablemente la generación de una animación. Sin embargo, es necesario mejorar la manera en la que se anima el avatar.

CAPÍTULO 2

2. Diseño e implementación

2.1 El usuario dibuja una pose

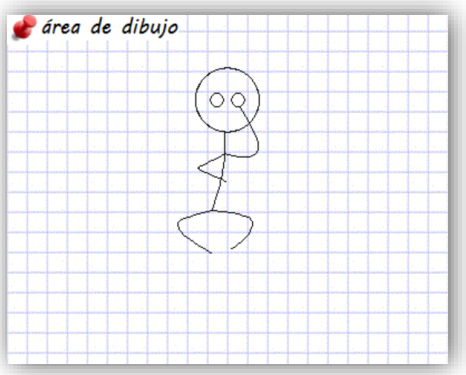


Figure 2.1 Área de dibujo del programa

En la figura 2.1 se muestra el área está destinada al dibujo de una pose para poder usarlo como una de las poses principales en una animación.

Desde el punto de vista del usuario, este debería dibujar a un stickfigure y este debe ser leído e interpretado. Sin embargo, hay algunas dificultades que el programa debe manejar por su cuenta, estas son:

- ¿Qué pasa si el pulso tembló un poco y la línea resultante no fue curva como el usuario esperaba?
- ¿Cómo se deberían guardar las poses de tal manera de que sea posible interpolar unas con otras?
- ¿Qué sucede si el usuario intenta dibujar algo que no es un ser humano?

En primer lugar, se definió una restricción para que el usuario no pueda dibujar una extremidad (contando con el torso) con más de un trazo o más de una extremidad de un solo trazo. Esta es una restricción relativamente sencilla y facilita muchos problemas de no tenerlo. Otra restricción que se hizo fue que el usuario siempre dibuje trazos conectados; es decir, no se puede dibujar un brazo suelto o una pierna suelta; siempre debe estar el torso dibujado primero. Además, se

definió que, independientemente de cuántos puntos genere el usuario al dibujar un trazo, éste se reduciría a 5 puntos y luego al mostrarse se usarían estos 5 puntos para generar una curva que pase por cada uno de ellos; de esta manera, no importa que tan poco pulso tenga el usuario, siempre dibujará una curva. Este paso no solo resuelve la primera pregunta sino que en parte la segunda, ya que siempre una pose tendrá el mismo número de puntos que otra; de esta manera se podrá interpolar punto por punto.

Pero para poder interpolar una pose a otra es necesario también conocer por cada punto de una pose cuál es su punto correspondiente en otra. Por esto es necesario conocer cuál extremidad es la que un usuario ha dibujado y en el caso de un brazo, si lo dibujó desde la mano hacia el hombro o viceversa (o desde el pie hacia la pelvis en el caso de una pierna). La manera en la cuál se identifica cada parte del cuerpo es simplemente identificando primero el torso como el primer trazo, los brazos como los trazos cuyo punto final o inicial está más cercano al segundo punto del torso (hombros) y las piernas los que están más cercanos a la parte final del torso (pelvis). Además, los puntos se reordenan de tal manera que siempre el primer punto del torso sea el que conecta a la cabeza y el primer punto de un brazo o una pierna sea el que conecta al torso.

Para que el usuario no pueda dibujar algo que no sea stickfigure, al empezar siempre tiene un círculo como guía; siendo este la representación de la cabeza con su respectivo par de ojos cuyo centro el usuario puede mover para simular la rotación de la cabeza. Además, se dibuja siempre 2 brazos y 2 piernas; si el usuario ya ha dibujado 2 brazos e intenta dibujar un trazo cercano al hombro, el programa automáticamente lo conecta a la parte de abajo del torso y lo vuelve una pierna. Cuando el usuario termina de dibujar los 5 trazos del stickfigure (piernas, brazos y torso) la parte de dibujo termina y no se le permite dibujar más trazos.

En esta etapa el usuario podría requerir modificar un poco el stickfigure una vez terminado, por lo cual puede ver estos 5 puntos en la pantalla y arrastrarlos para encontrar la figura que desea.

2.2 El programa hace los frames de relleno

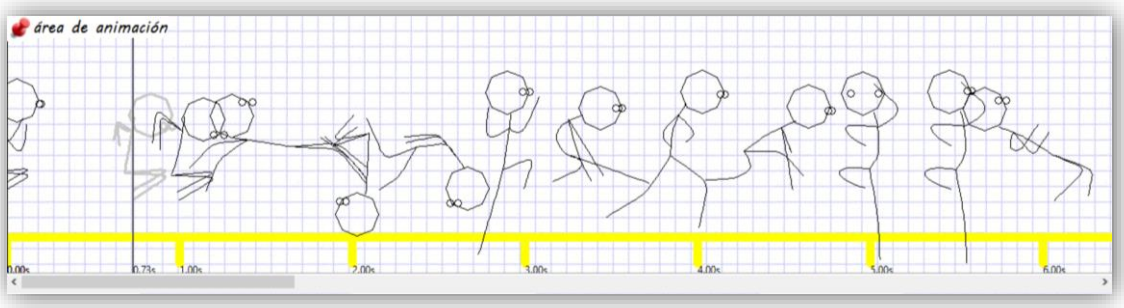


Figura 2.2: Área de animación del programa.

En la figura 2.2 se muestra el área de la animación del programa, aquí se muestra una línea de tiempo que va de 0 a 6 segundos. Además, se pueden observar las poses principales dibujadas en color negro y las poses intermedias, de color gris claro. Éstas últimas se muestran sólo en la posición de la barra vertical que hace las veces de puntero seleccionando el tiempo que queremos señalar.

Antes de explicar cómo el programa interpola las poses, cabe mencionar que esta área de animación de por sí también tiene sus dificultades intrínsecas como por ejemplo el hecho de que; por poner un ejemplo, si un usuario quiere modificar una pose que está en el segundo 2,56:

- ¿Cómo le hace para seleccionar dicho segundo y no el 2.57 o 2.5671?
- ¿Cuántas poses intermedias se debería generar?
- ¿Cuántas son muchas?
- ¿Cuántas son muy pocas?

La solución a todos estos conflictos fue el restringir a que haya 30 poses por cada segundo; esto quiere decir que siempre que el usuario haga click en cierto lugar de la línea del tiempo seleccionará un segundo múltiplo de 1/30 y nunca se equivocará al seleccionar un frame.

En esta etapa el programa tiene todas las poses que el usuario ha dibujado, cada pose es un conjunto de puntos y el programa puede interpolar punto por punto de cada pose. La primera solución que nos llega a la mente es interpolar linealmente cada punto; es decir; por poner un ejemplo, si el punto de la mano de la primera pose es [0,0] y el punto de la mano de la segunda es [10,10];

entonces, las manos de las poses intermedias tendrían como valor [1,1], [2,2], [3,3],….[9,9].

Sin embargo esta es una mala solución ya que al momento de reproducir esta animación se ve muy poco natural.

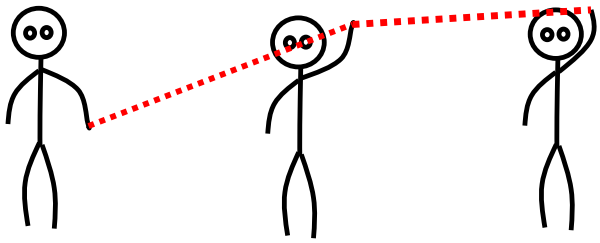


Figura 2.3: Mano de una pose interpolada linealmente.

En la figura 2.3 se intenta representar a las manos como los puntos que están al final de cada brazo; siendo los puntos rojos, puntos interpolados linealmente de estas manos en las poses intermedias entre estas tres poses principales.

La solución que se encontró fue interpolar los puntos con curvas en vez de interpolarlos linealmente; esto hizo que la animación se viera mucho mejor. Sin embargo, estas curvas no pasan exactamente por los puntos de las poses principales, sino cerca.

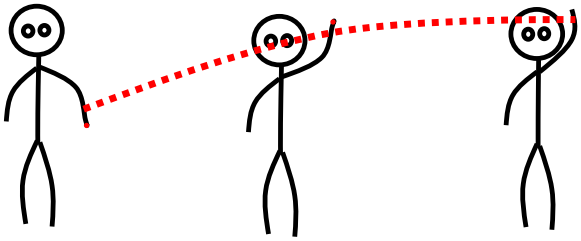


Figura 2.4: Mano de una pose interpolada por una curva spline.

En la figura 2.4 se intenta representar la misma interpolación de la imagen anterior; pero hecha a manera de una curva spline. Cabe notar que el punto no pasa exactamente por la mano; sin embargo, esta es una imprecisión poco perceptible.

2.3 La pose mueve al avatar

En esta parte cada pose debe transformarse a 3D y luego girar los joints (articulaciones) del avatar cargado para poder animarlo. Para hacerlo, primero se encuentra la componente de profundidad en cada punto; esto se lo hace con proporciones definidas en el código que hacen posible inferir la longitud máxima que debería tener cierta parte, y con ciertas suposiciones de cómo se mueve cada parte. Por ejemplo, con respecto al torso y los brazos, se asume que si hay una profundidad será hundiendo sus codos hacia dentro de la pantalla o su vientre en el caso el torso, y en el caso de las piernas moviéndolas hacia afuera de la pantalla.

En este paso se asume que el brazo izquierdo está a la izquierda del torso y el brazo derecho a la derecha del torso. Lo mismo se asume para las piernas; caso contrario, si la profundidad de dicha parte es hacia afuera de la pantalla se la invierte hacia adentro y viceversa.

Una vez que tenemos esta pose debemos adaptarlas al esqueleto del avatar en 3D y también adaptar el esqueleto del avatar a la pose.

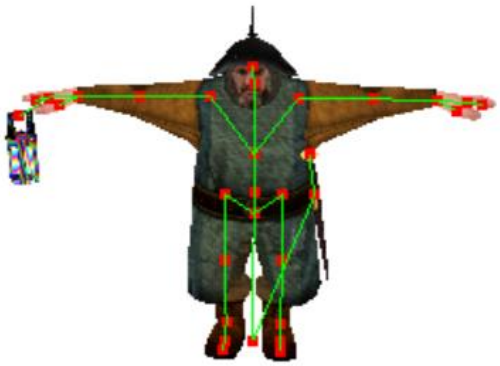


Figura 2.5: Avatar en 3D

En la figura 2.5 vemos un avatar que tiene un farol en una mano y una espada envainada a su izquierda; éste, como muchos otros avatares en 3D no es homogéneo ni simétrico, así que habrá que encontrar sus extremidades y su torso antes de poder animarlo.

Para esto se separa cada joint en grupos. Cada grupo es un conjunto de puntos que está unido entre sí en una línea; una vez que un punto se bifurca pertenece a otro grupo.

Esto da como resultado un conjunto de grupos desde los cuales se seleccionan a los 5 grupos más grandes y se identifica al torso, cuyo centro está en medio de los 5; los brazos como los grupos cuyo centro está más arriba y las piernas como las partes cuyo centro están más abajo. De manera similar se identifican la pierna y el brazo izquierdo como las que tienen el centro más a la izquierda y la pierna y el brazo derecho como las que tienen el centro más a la derecha.

Y una vez más se ordenan los puntos para que el primero del torso sea el que esté conectado a la cabeza, y el primero de los brazos y piernas sean los que estén conectados al torso.

Adicionalmente se buscan los joints que corresponden a los pies y se los elimina. Luego se realiza una representación que modifica los puntos para que los brazos y las piernas siempre estén rectas.

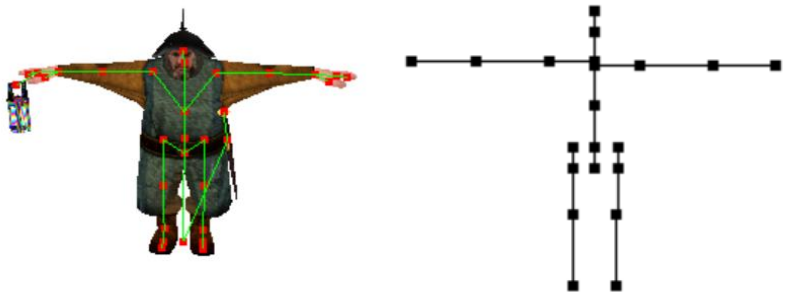


Figura 2.6: Avatar en 3D junto a la representación de su esqueleto

En la figura 2.6 se puede ver el esqueleto del mesh junto a su representación; en este paso los brazos se vuelven completamente horizontales y las piernas verticales.

Una vez hecho esto hace falta adaptar la pose al esqueleto identificado. La pose tiene invariablemente cinco puntos por parte, mientras que éste esqueleto en específico tendría tres por brazo, (cinco en el torso y cinco en las piernas). Así que se hace una curva de 30 puntos por cada parte y luego se seleccionan los puntos que corresponden al esqueleto del avatar.

Sin embargo, antes de continuar con la explicación es conveniente explicar el concepto de un ente matemático requerido en esta parte del proyecto:

2.4 Los cuaterniones

Los cuaterniones son una representación de una rotación en 3 dimensiones, formada por un número real y un vector imaginario de 3 dimensiones:

A+bi+jc+dk.

Esta representación no solamente ayuda a que la rotación sea de manera más suave, sino que también previene el problema de Gimbal-Lock que sucede cuando se gira un objeto con respecto a sus ejes x, y y z; llegando un momento en el cual pierde uno o dos grados de libertad[1].

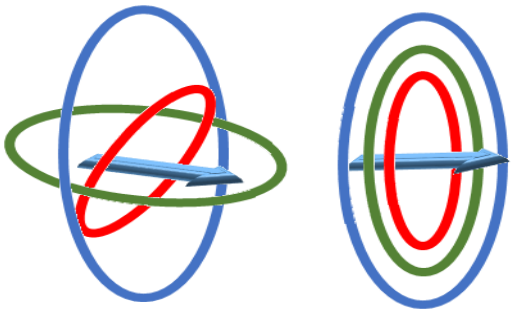


Figura 2.7: ilustración del Gimbal Lock

En la figura 2.7 se puede apreciar el problema conocido como Gimbal Lock. La flecha puede moverse con respecto a su eje x, y y z locales libremente, sin embargo una vez que dos o más ejes coinciden se pierde la capacidad de rotarlo en algún eje[2].

Aunque hay muchas operaciones que se pueden realizar con cuaterniones, para efectos de este proyecto se utilizaron 2 en especial: la multiplicación entre cuaterniones y la conversión desde un ángulo y un eje a un cuaternión.

Ahora es necesario explicar de qué está hecho un Joint. Cada joint está hecho de un punto y un cuaternión: el punto representa el lugar donde está el objeto representado por dicho joint, mientras que cada cuaternión representa la orientación de la parte del objeto 3D que está siendo controlado por dicho joint.

Figura 2.8: movimiento del joint de la cabeza

De izquierda a derecha las imágenes representan:

- La cabeza en su posición y orientación original.
- El punto de la cabeza rotada con respecto al punto de su joint padre (el cuello) con respecto al eje Y (vector hacia adentro de la pantalla).
- El cuaternión de la cabeza rotado con respecto al eje Y.
- El joint de la cabeza rotado con respecto al eje Y tanto el punto como la orientación.

Esto significa que para rotar un joint debemos de rotar independientemente su punto y su cuaternión.

Por último, para poder aplicar la pose al esqueleto se calcula los ángulos entre los joints; esto es, el ángulo de un joint con otro con respecto al eje x, y y z.

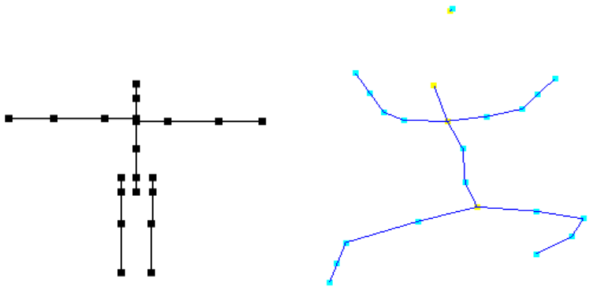


Figura 2.9: representación del esqueleto junto a la pose en 3D

En esta figura se muestra a la izquierda la representación del esqueleto del avatar y a la derecha la pose dibujada en el programa. Por cada punto de dicha representación se calcula el vector que viene desde su joint padre hasta el actual y se obtienen los ángulos con respecto a su respectivo joint en la pose; de esta manera se sabe cuánto se debe rotar al joint del esqueleto de nuestro avatar para que tenga la misma orientación.

Primero se gira cada joint con respecto al eje z, luego con respecto al eje y y por ultimo con respecto al eje x.

En el caso de los ojos, la posición de los ojos está guardado en un punto 2D cuyos valores van de -1 a 1, de tal manera que 0,0 significa que la cabeza está viendo al frente, -1,-1 que está viendo hacia abajo a la izquierda y 1,1 que está viendo hacia arriba a la derecha.

CAPÍTULO 3

3. Resultados.

3.1 Área de dibujo

En esta área se optó por medir la usabilidad tomando en cuenta la percepción de dificultad que tiene el usuario al dibujar las poses con valores del 1 al 10 y el tiempo que se demoró en dibujar cada pose. En esta prueba se tienen 3 poses: A, B y C.

Las poses se muestran en la figura 3.1, la tercera pose es una pose inventada por el usuario.

Figura 3.1: poses de prueba

En la figura 3.1 se muestran las poses de prueba a usarse para la encuesta. La primera pose está pensada para ser sencilla, la segunda pose está pensada para ser un poco más compleja que la primera, mientras que la tercera pose es una pose que el usuario pueda inventar.

A continuación los datos obtenidos:

		Usuario									
Pose		1)	2)	3)	4)	5)	6)	7)	8)	9)	10)
	a)	19	12	27	15	10	10	12	10	17	17
	b)	22	10	15	15	17	21	17	18	17	23
	c)	34	7	17	6	36	12	14	10	13	18

Tabla 1: Tiempo necesario para hacer cada pose.

En la tabla 1 se muestra el tiempo requerido en segundos para dibujar las poses por cada usuario.

		Usuario									
Dificultad		1)	2)	3)	4)	5)	6)	7)	8)	9)	10)
	a)	1	2	1	1	1	1	3	2	1	1
	b)	1	3	2	1	1	1	3	2	2	1
	c)	3	2	1	1	1	1	2	1	1	1

Tabla 2: Índices de dificultad percibida.

En la tabla 2 se muestra la dificultad que cada usuario percibió para dibujar cada pose siendo 1 muy fácil y 10 muy difícil.

3.2 Área de animación

En esta área se midió la percepción de los usuarios frente a la animación resultante de estas 3 poses, la pregunta era: ¿la animación fue la que se esperaba? Donde 1 significa completamente en desacuerdo y 10 completamente de acuerdo.

USUARIO	CALIFICACIÓN
1	9
2	8
3	10
4	9
5	10
6	2
7	8
8	9
9	10

10	10

Tabla 3: Calificación de la animación

En la tabla 3 se muestra la calificación de la calidad de la animación dada por cada usuario, donde 1 significa calidad pobre y 10 alta calidad.

3.3 Área del avatar

En esta área se midió la percepción de los usuarios frente a la animación aplicada al avatar, donde 1 significa que se ve completamente mal y 10 completamente bien.

USUARIO	CALIFICACIÓN
1	6
2	3
3	8
4	3
5	5
6	2
7	5
8	5
9	7
10	8

Tabla 4: Calificación de la animación del avatar

En la tabla 4 se muestra la calificación de la calidad de la animación aplicada al avatar dada por cada usuario, donde 1 significa calidad pobre y 10 alta calidad.

3.4 Análisis de los datos obtenidos

En el área de dibujo el tiempo promedio requerido para dibujar las poses fueron los siguientes:

a) 14.9 +/- 5.4s
b) 17.5 +/- 3.8s
c) 16.7 +/- 10.4s

Lo cual se explica teniendo en mente que la primera pose fue pensada para ser una pose sencilla, la segunda un poco más difícil y la tercera varió bastante dependiendo de qué tanta dificultad tenía la pose que el usuario quería dibujar.

Por otro lado la dificultad percibida promedio para dibujar las poses fueron las siguientes:

a) 1.4 +/- 0.7
b) 1.7 +/- 0.8
c) 1.4 +/- 0.7

Lo cual indica que en general al usuario le pareció sencillo de dibujar en esta aplicación.

En el área de animación se calificó la calidad de esta con un promedio de 8.5 +/- 2.4 que significa muy bueno.

Sin embargo en el área del avatar esta calificación bajó a 5.2 +/- 2.1 en promedio lo cual indica que es necesario mejorar en ese aspecto.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

1. Este proyecto ha permitido hacer de manera más sencilla y natural la animación de un stickfigure.
2. Este proyecto también ha permitido adaptar la animación a un avatar en 3D.
3. Sin embargo hay aspectos que considerar antes de modificar un esqueleto de un avatar en 3D. Uno de estos es el hecho de que cada j oint tiene un punto y un cuaternión; hay que modificar ambos independientemente para que giren correctamente.

RECOMENDACIONES

1. Familiarizarse con los cuaterniones antes de trabajar con animaciones en 3D.
2. A veces en algunos tutoriales se usa el formato [w,x,y,z] para expresar los cuaterniones, en otros sin embargo, se utiliza [x,y,z,w]; es necesario identificar donde consideran la parte real (w) para evitar errores.
3. Encontrar un formato de archivos en 3D que sea sencillo de leer y manejar.
4. Enfocarse en la facilidad del usuario para manejar el programa.
5. Es una buena idea mejorar el programa permitiéndole importar más formatos de archivos 3D.
6. El programa puede ser mejorado al permitirse guardar el nuevo esqueleto generado.

BIBLIOGRAFÍA

[1] “hexapod@netcom.com”, (1998,Diciembre,26) The Matrix and Quaternions FAQ. Disponible en: http://www.flipcode.com/documents/matrfaq.html#Q47

[2] Cyrille Fauvel, (2012, Agosto, 10) Avoid Gimbal Lock for Rotation/Direction Maya Manipulators [online]. Disponible en: http://around-the-corner.typepad.com/adn/2012/08/avoid-gimbal-lock-for-rotationdirection-maya-manipulators.html

ANEXOS

[1] Jeremiah Van Oosten (2012,Junio,25) Understanding Quaternions [online]. Disponible en: http://www.3dgep.com/understanding-quaternions/

[2] Cuestionario del proyecto

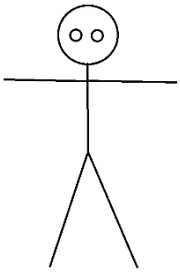
[3] PseudocódigoStick

Encuesta de usabilidad

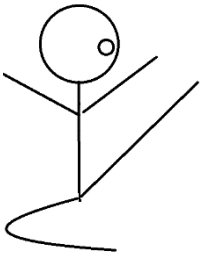
Parte de dibujo

Dibuje las siguientes poses

a)



b)



c)

Dibujar una pose cualquiera

Tiempo que se demoró:

a) _____

b) _____

c) _____

Dificultad para dibujar la pose: (un valor del 1 al 10, siendo 1 muy fácil y 10 muy difícil)

a) _____

b) _____

c) _____

Parte de animación:

¿Dada las poses principales dibujadas, es la animación que se esperaba? (responder con un valor del 1 al 10, siendo 1 animación de calidad pobre y 10 animación de buena calidad)

_____

Parte del avatar:

¿Dada las animación resultante, el avatar se movió como se esperaba? (responder con un valor del 1 al 10, siendo 1 bastante mal y 10 bastante bien)

Pseudocódigos de StickAnimator

Algoritmo de generación de Stickfigure

Una vez que el usuario dibuja el stickfigure es necesario ordenarlo para que pueda ser manipulado junto a otras poses. Esto se lo hace con la función mostrada a continuación:

DISTANCE(p1,p2): es la distancia entre p1 y p2.

CENTROID(points): obtiene el punto medio de todos los puntos en el arreglo enviado como parámetro.

Strokes es un arreglo de strokes y cada stroke es un arreglo de puntos. Esta función asume además que el primer stroke de la lista es el torso y que cada stroke está ordenado de tal manera que el primer punto de cada stroke está unido al torso.

ORDENAR_BODY(strokes)

```
1   brazos←[]
2   newstrokes←[strokes0]
3   jointArms←strokes01
4   jointLegs←strokes0 |strokes0| //el punto final del torso
5   for i from 1 to |strokes|-1 do
6     distshoulder←DISTANCE(jointArms,strokesi 0)
7     distpelvis← DISTANCE(jointArms,strokesi (|strokes i|-1))
8     distancearms←MIN(distshoulder,distpelvis)
9
10    distshoulder←DISTANCE(jointLegs,strokesi 0)
11    distpelvis← DISTANCE(jointLegs,strokesi (|strokes i|-1) )
12    distancelegs←MIN(distshoulder,distpelvis)
13    If distancearms<distancelegs then
14      APPEND(brazos,i)
15    If |brazos|=2 then
16      break
17  //aquí nos aseguramos que el brazo izquierdo esté antes en la lista
18  If CENTROID(strokes[brazos0])x>CENTROID(strokes[brazos1])x then
19    REVERSE(brazos)
20  APPEND(newstrokes,strokes[brazos0])
21  APPEND(newstrokes,strokes[brazos1])
22  indexes←[1,2,3,4]
23  REMOVE(indexes,brazos0)
```

```
24    REMOVE(indexes,brazos1)//nos aseguramos que solo nos queden los
      índices de las piernas
25    //aquí nos aseguramos que la pierna izquierda vaya primero
26    If CENTROID(strokes[indexes0])x> CENTROID(strokes[indexes1])x then
27       REVERSE(indexes)
28    APPEND(newstrokes,strokes[indexes0])
29    APPEND(newstrokes,strokes[indexes1])
30    return newstrokes
```

En la función anterior se asume que todos los strokes están ordenados de tal manera que el primer punto es el que va unido al torso. Esto se lo hace con la función mostrada más adelante que recibe el stroke que el usuario ha dibujado y retorna el stroke que es el cual finalmente se guarda.

Esta función además une los brazos y las piernas con el torso y el torso con la cabeza y limita el número de piernas y de brazos a dos.

Esta función asume además que hay una cabeza (“head”) que es simplemente los puntos de la cabeza que se genera por defecto cada vez que se comienza a dibujar un stickfigure.

COMPLETE_POINTS(points,strokes,head)

```
1     np←points[:]//se copia los puntos de points a np
2     //esta parte se ejecuta si el stroke es el torso
3     if |strokes|=0 then
4        ch←CENTROID(head)
5        If DISTANCE(ch, np|np|-1)<=DISTANCE(ch, np0) then
6           REVERSE(np)
7        np←ARREGLAR_CUELLO(np,head)
8        return np
9     jointArms←strokes01
10    jointLegs←strokes0 |strokes 0|-1
11    distancearms←MIN(
12    DISTANCE(jointArms,np0),DISTANCE (jointArms, np|np|-1))
13    distancelegs←MIN(
14    DISTANCE(jointLegs, np0),DISTANCE(jointLegs,np|np|-1))
15    if(NUMPIERNAS(strokes)>=2
16    or (distancearms<distancelegs and NUMBRAZOS(strokes)<2)) then
17       if DISTANCE(jointArms,np0)>DISTANCE(jointArms,np|np|-1)then
18          REVERSE(np)
19       INSERT(np,jointArms)
20    else
21       If DISTANCE(jointLegs,np0)>DISTANCE(jointArms,np|np|-1)then
22          REVERSE(np)
23       INSERT(np,jointLegs)
```

```
24      return np
```

ARREGLAR_CUELLO(points,head)

```
1       np←points[:]
2       c←CENTROID(head)
3       rad←DISTANCE(head0,head|head|/2)/2
4       while |np|>0 and ESTADENTROCABEZA(np0,c,rad)then
5          np←np[1:] //se elimina el primer punto de np
6       INSERT(np,0,GETCUELLO(np,head))
7       return np
```

ESTADENTROCABEZA(p,c,rad)

```
1       return DISTANCE(p,c)<=rad
```

GETCUELLO(points,head)

```
1       c←CENTROID(head)
2       rad← DISTANCE(head0,head|head|/2)/2
3       start←points0
4       end←c
5       delta←[endx-startx,endy-starty]
6       p←1-rad/DISTANCE(start,end)
7       end←[startx+p*deltax,starty+p*deltay]
8       return end
```

NUMBRAZOS(strokes)

```
1       num←0
2       jointArms←strokes01
3       jointLegs←strokes0 (|strokes|-1)
4       for i from 1 to |strokes|-1 do
5          distancearms←MIN(
6          DISTANCE(jointArms,strokesi0),
7          DISTANCE(jointArms,strokesi |strokes i|-1))
8          distancelegs←MIN(
9          DISTANCE(jointLegs,strokesi0),
10         DISTANCE(jointLegs,strokesi |strokes i|))
11         If distancearms<distancelegs then
12            num←num+1
13      return num
```

NUMPIERNAS(strokes)

```
1       return |strokes|-NUMBRAZOS(strokes)-1
```

Pseudocódigo del algoritmo de interpolación

VECTORIZE(pose): transforma una pose del stack en un arreglo de puntos.

UNVECTORIZE(puntos): transforma un conjunto de puntos de vuelta en una pose.

BSPLINE(puntos,K,num): genera una curva dado el conjunto de puntos enviado como parámetro, un valor de coeficiente K, y el número de puntos resultante

RESAMPLED(puntos,num): genera un conjunto de puntos que tiene la misma trayectoria que el conjunto de puntos enviado como parámetro, dado un conjunto de puntos y el número de puntos enviado como parámetro

Stack: arreglo de poses principales. Cada pose principal es un arreglo de "partes" (cabeza, torso y extremidades), y cada parte es un arreglo de puntos. El stack representa a todos los frames presentes en la animación, sean estos frames vacíos o poses principales. Los frames vacíos, aquellos que no tienen poses, se representan con un arreglo vacío.

Num: número de poses intermedias a generarse entre cada pose principal. Mientras más cantidad, mejor calidad, pero toma más tiempo.

CURVESPLINE(stack,num)

```
1   interp←[ ]
2   foreach p in stack do
3      if |p|!=0 then
4         APPEND(interp,VECTORIZE(p))
5
6   //si hay menos de 3 poses principales, se interpola linealmente.
7   If |interp|<=2 then
8      return stack
9
10  n←|interp|
11  dim←|interp0|
12  newint←[interp0]
13
14  //se agrega una pose intermedia entre cada par de poses principales con
15  la finalidad de obtener un resultado con mejor calidad.
16  for i from 1 to |interp|-1 do
17     point←[ ]
18     for j from 0 to |dim|-1 do
19        APPEND(point,(interpi j+interpi-1 j)/2)
20     APPEND(newint,point)
21     APPEND(newint,interp[i])
22
23  Interp←newint
24  K←3
25  flen←((n-1)*num)+n
26  interp_array←[ ]
27
28  // se genera una curva por cada par de poses principales que tiene "flen"
```

```
29      número de puntos
30
31      for i from 0 to |dim|-1 do
32          points←interp
33          x←BSPLINE(points[:,i], K , flen)
34          APPEND(interp_array,x)
35
36      init←0
37      end←-1
38      n←0
39      resampled←[]
40      bet←0
41      for i from 1 to |self.stack|-1 do
42          if |self.stack_i|=0 then
43              n←n+1
44          else
45              INSERT(resampled, interp_array_(num+1)*bet)
46              end←i
47              If n>1 then
48                  resampled+=RESAMPLE(
49                  interp_array[1+((num+1)*bet):(num+1)*(bet+1)],n+2)[1:-1]
50              else
51                  if(n=1) then
52                      INSERT(
53                      resampled,
54                      interp_array[int((1+((num+1)*bet)+(num+1)*(bet+1))/2)]])
55              bet←bet+1
56              init←i
57              n←0
58      INSERT(resampled,[interp_array[-1]])
59      self.interpoled←[ ]
60      foreach v in resampled do
61          APPEND(
62          interpoled,UNVECTORIZE(v))
63      return interpoled
```

Multiplicación entre cuaterniones

Q_MULT (Q1, Q2)

```
1       Qr←[0,0,0,0]
2       Qr_x←Q1_x*Q2_x- Q1y*Q2_y - Q1_z*Q2_z - Q1_w*Q2_w
```

```
3       Qry←Q1x*Q2y+ Q1y*Q2x + Q1z*Q2w - Q1w*Q2z
4       Qrz←Q1x*Q2z- Q1y*Q2w + Q1z*Q2x - Q1w*Q2y
5       Qrw←Q1x*Q2w+ Q1y*Q2z - Q1z*Q2y + Q1w*Q2x
6       return Qr
```

[1]

Convertir un ángulo y un eje a un cuaternión

Q_FROM_V_THETA (V, theta)

```
1       S←sin(theta/2)
2       C←cos(theta/2)
3       Vnrm← MODULE(V) // obtiene el módulo de un vector
4       Q←CONCATENATE(s*v/vnrm, [c])
5       return Q
```

[2]

//esta función rota el cuaternión de un joint

ROTATE_Q_GLOBAL (joint, axis, angle)

```
1       quat←joint.Orient
2       rotmin← Q_FROM_V_THETA(axis,angle)
3       joint.Orient← Q_MULT(CONJUGATE(quat),CONJUGATE(rotmin))
4       joint.Orient←CONJUGATE(joint.Orient)
```

CONJUGATE (Q)

```
1       return [-Qx,-Qy,-Qz,Qw]
```

//En la siguiente función rotamos el punto del joint con respecto al punto del joint padre

MOVE_POINT_GLOBAL (joint, axis, angle)

```
1       parentJoint←joint.Parent //obtiene el joint padre
2       pp←parentJoint.Pos //obtiene la posición del Joint padre
3       fp←joint.Pos// obtiene la posición del Joint actual
4       deltapoint←[fpx-ppx, fpy-ppy, fpz-ppz]
5       rot← Q_FROM_V_THETA(axis,angle)
6
7       //rota un vector dado un cuaternión
8       deltapoint←VECTOR_Q(deltapoint, rot)
9       joint.Pos←[ppx+deltapointx, ppy+deltapointy, ppz+deltapointz]
```

VECTOR_Q (v, q):

```
1       u←[qx, qy,qz]
2       s ← qw
```

3 **return** u*(2*DOT(u,v)) + v*(s*s- DOT (u,u)) + CROSS(u,v)*(2*s)

DOT (v1, v2) calcula el producto punto entre 2 vectores

CROSS (v1, v2) calcula el producto cruz entre 2 vectores

BIBLIOGRAFÍA

[1] “Confuted”, Using Quaternion to Perform 3D rotations [online]. Disponible en: http://www.cprogramming.com/tutorial/3d/quaternions.html

[2] Jake Vanderplas, (24, Noviembre, 2012), Quaternions and Key Bindings: Simple 3D Visualization in Matplotlib [online]. Disponible en: https://jakevdp.github.io/blog/2012/11/24/simple-3d-visualization-in-matplotlib/