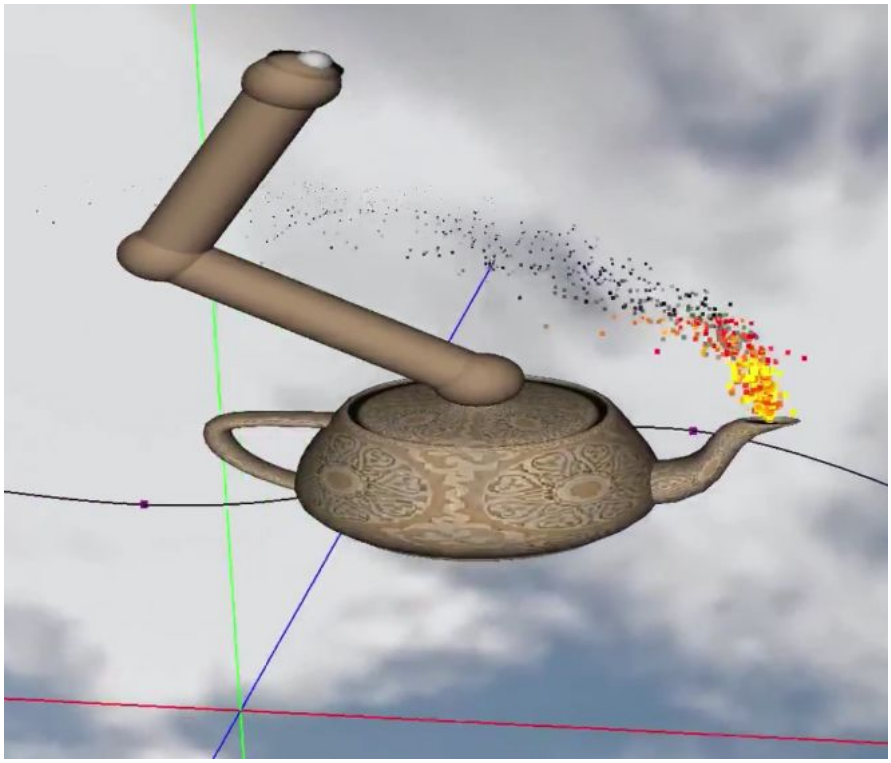
Curso 2015-2016

Técnicas de Animación 3D y post-procesamiento

Práctica 8:
ANIMACIÓN COMPLETA



Nombre: Daniel Martínez Caballero
DNI: 77375733F
Correo: dmc00024@red.ujaen.es

Documentación y manual de usuario:

Para la realización de esta aplicación se ha usado como base la estructura empleada en la asignatura Informática Gráfica y Visualización, biblioteca GLut.
La velocidad en frames empleada se encuentra limitada a 30 FPS desde código para que la velocidad de procesamiento de cada equipo no afecte en su ejecución.

Debido a la dificultad para construir una interfaz gráfica con la biblioteca GLut se procederá a modificar todos los datos de entrada desde los ficheros: **keyframes.txt, traveling.txt y speedCurve.txt**. Estos ficheros deberán situarse en la carpeta del proyecto o bien del ejecutable para que sean cargados correctamente, de lo contrario, no nos mostrará nada la escena.
Las funciones encargadas de cargar estos ficheros en el programa estarán situadas en *"Util.h"*.

El fichero '*keyframes.txt'* es necesario para ajustar la traslación del objeto por la escena y *'traveling.txt'* de un mismo modo para el viaje de la cámara.
Ambos ficheros tendrá la misma estructura: número de fotogramas clave, coordenadas tridimensionales de estos fotogramas y fotogramas de referencia.

Ejemplo:

```
3
-10 -5 5
3 10 15
9 2 -5
1000
5000
10000
```

Técnicas de Interpolación:

-Si la opción de interpolación lineal se encuentra activada, ver manual de teclas, a estos puntos se les realizará una **interpolación lineal** (LERP) de forma automática. De tal modo que el objeto los recorrerá en los puntos establecidos tomando como referencia los frames correspondientes del fichero de entrada.

Para ver el código de esta sección ir al archivo *"igvEscena3D.cpp"* en el método de "*interpolacionLineal()"*.
Aquí encontraremos el dibujado de la trayectoria de todos los fotogramas clave y el fragmento de código que se encargará de ir aplicando la interpolación al objeto en cada frame de la ejecución.
La función '*makeLerp(. . .)'* situada en el fichero *"Interpolation.h"* se encargará de realizar la interpolación lineal en los tres ejes siguiendo la fórmula:

```
LERP = (1 - alpha) * p1 + p2 * alpha;
```

- **Para la implementación de la interpolación esférica** ha sido necesaria la implementación de la clase Quaternions y sus operaciones.
Por defecto se realizará una rotación en los ejes 'Y' y 'Z' a partir del 20% *(lambda)* del recorrido entre dos puntos, de modo que el objeto termina esta interpolación orientándose con respecto a la siguiente recta en el eje 'Y'.

El SLERP ha sido configurado por constantes en el código, si se quiere personalizar debemos ir al fichero *'igvEscena3D.cpp'* y en el método de '*interpolacionEsferica()'* tendremos todo lo referente a esta. Si queremos modificar el porcentaje en el cual el objeto comienza a realizar esta interpolación debemos modificar el valor de *'rate'* entre 0 y 1.

Para ver la implementación de las operaciones necesarias en las interpolaciones debemos ir a la función *'makeSlerp(...)'* en el fichero *'Interpolation.h'*. En esta interfaz se sitúa la definición de la clase Point3D, Quaternion y funciones que nos sirven de utilidad a la hora de realizar el SLERP.

- **El escalado no uniforme** que se realiza por defecto es una pequeña contracción del objeto en las curvas.

Para modificar el escalado no uniforme debemos ir en el mismo fichero de la escena, en la función llamada "*escaladoNoUniforme()"*. Aquí podemos modificar el valor de '*rateScale'* el cual nos permitirá (al igual que el rate de la interpolación esférica) establecer el punto de partida del escalado no uniforme.
El escalado que se realiza se ha representado con la estructura de *'Point3D'* y va desde 'r1' a 'r2' realizando una interpolación.

- **El traveling de cámara** a nivel de implementación funciona igual que la interpolación del objeto pero usando el fichero '*traveling.txt*'. Además, mantendremos el punto de interés fijado en el objeto en todo momento.
Para ello, en el fichero '*igvInterfaz.cpp*' en la línea 165 tendremos el fragmento de código que trabajará para realizar el traveling correctamente.

Manejo de curvas: Interpolación de Hermite

- A estos puntos se les realizará una **interpolación curva** (Interpolación de Hermite) por defecto al arrancar la aplicación. De tal modo que el objeto los recorrerá en los puntos establecidos tomando como referencia los frames correspondientes del fichero de entrada.

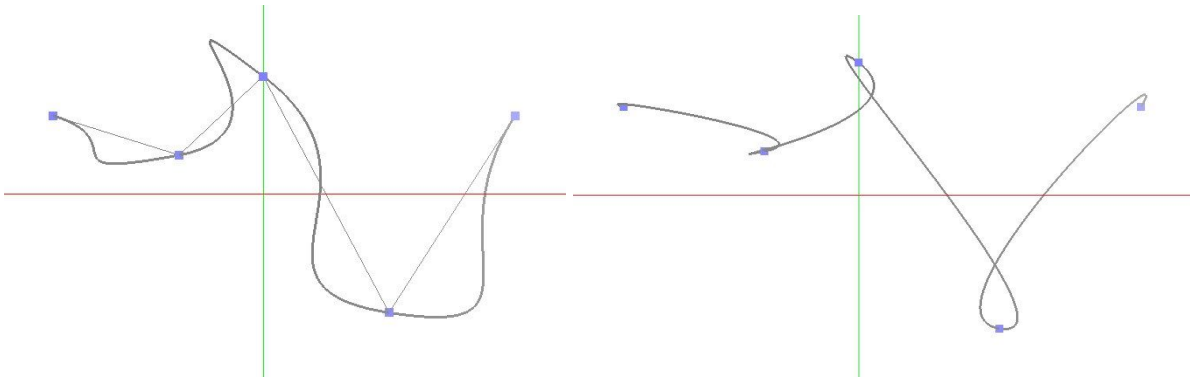
Para ver el código de esta sección ir al archivo *"igvEscena3D.cpp"* en el método de "*interpolacionCurva()".*
Aquí encontraremos el dibujado de la trayectoria mediante un buffer y el fragmento de código que se encargará de ir aplicando la interpolación al objeto en cada frame de la ejecución.
La función '*HermiteInterpolate(. . .)*' situada en el fichero *"Interpolation.h"* se encargará de realizar la interpolación curva en los tres ejes.
Esta función polinómica realizará una interpolación entre cuatro puntos datos: el punto anterior al origen, punto origen, punto destino y punto sucesor al destino.
De este modo la función de Hermite necesita saber tanto los puntos de control entre los que se encontrará, origen y destino, como el anterior y el siguiente respectivamente.
Con este tipo de interpolación surge un problema que se deberá solucionar de algún modo u otro, esto ocurre cuando queremos interpolar entre los primeros puntos '0 a 1' y cuando lo hacemos con los últimos 'n-1 a n'.
Existen varias alternativas pero la decisión final ha sido la de tomar como puntos ficticios el propio inicio o final quedando así:

- Para el primer caso: '0, 0, 1, 2'.
- Para el último caso: 'n-2, n-1, n, n'.

Quedando así solucionado el problema de una forma sencilla.

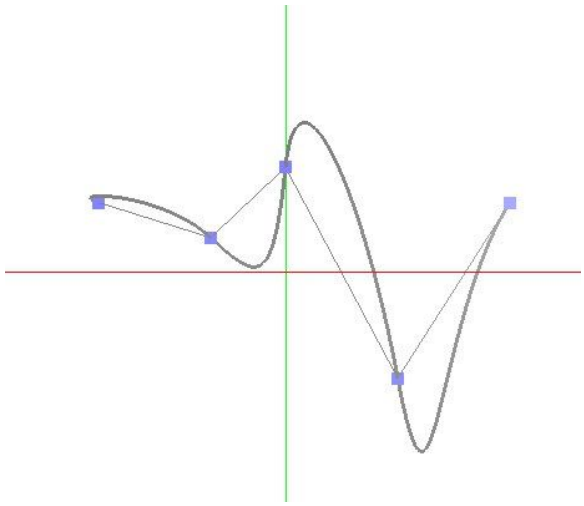
-Uno de los motivos por los que se ha escogido la interpolación cúbica de Hermite, aparte de su sencillez, es la posibilidad de poder modificar el tipo de curva que queremos generar.

El primer parámetro es la ‘Tensión’, el cual se recomienda modificar entre los valores 1 y -1. Con la tensión con valor de 1 conseguiremos una interpolación lineal, esto es debido a que estaríamos maximizando la tensión entre los puntos y por lo tanto es lógico que ocurra. En el caso contrario, si se usan valores negativos la curvatura será más exagerada, como si fuera una cuerda floja. *(imagen 1)*



Como curiosidad, si se establece un valor mayor que 1 para la tensión ocurre este efecto de “nudo” *(imagen 2).*

Otro de sus argumentos es ‘Bias’, este nos puede ser útil si queremos adelantar o retrasar el eje de la curvatura con respecto al punto clave correspondiente.



-El escalado no uniforme y las rotaciones SLERP seguirán funcionando como en la práctica anterior con pequeños ajustes para acompañar a la interpolación curva.

- **El traveling de cámara** usará también la interpolación de Hermite con el punto de interés fijado en el objeto en todo momento.
Para ello, en el fichero '*igvInterfaz.cpp*' en la línea 165 tendremos el fragmento de código que trabajará para realizar el traveling correctamente.

- **Curva de velocidad** se ha tratado de implementar una curva de velocidad modificable desde el fichero *'speedCurve.txt'* aquí encontraremos una serie de valores que se explican a continuación:

```
3
0.1
0.3
0.8
0
0.2
1
```

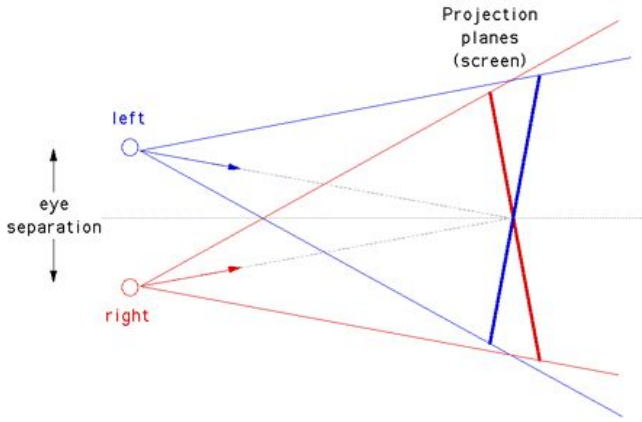
En la primera línea se indicará el número de puntos de control.
Los primeros n-valores será el multiplicador de velocidad, un valor entre 0 y 1.
El siguiente conjunto de valores indicará la distancia en ratio donde se aplicarán los puntos de control para así interpolar las velocidades.

Para explorar con detalle esta opción ir al fichero *'igvEscena3D.cpp'* en el método de Interpolación Curva encontraremos un segmento de código con las operaciones utilizadas para el desarrollo de esta utilidad.

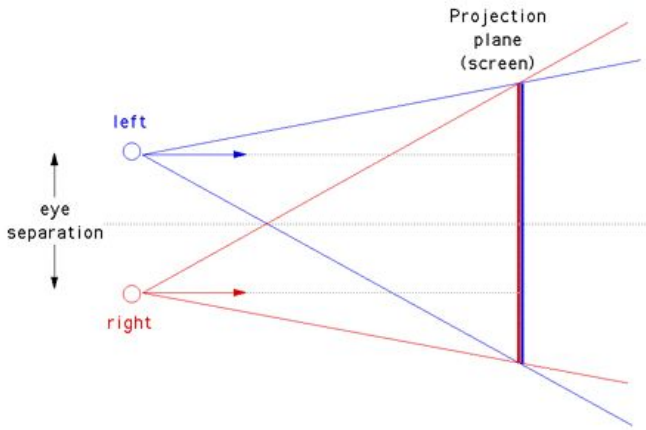
Estereoscopía: Par estéreo

Lo siguiente a explicar es la implementación de la cámara estereoscópica virtual, es decir, un par estéreo que nos ofrezca una sensación de profundidad al visualizar en diferentes modos de 3D (anaglifo, polarización, cardboards...etc).

Existen dos métodos para ajustar una cámara virtual y renderizar en dos imágenes estéreo. Muchos de estos métodos son incorrectos ya que se produce paralaje vertical, esto significa que los planos proyectados por cada una de las cámaras se muestran una disparidad fuera del centro de los ejes.
Un ejemplo de un método incorrecto es el llamado Toe-in, el cual establece como punto de interés el mismo punto, creando planos cruzados tal y como se muestra en la imagen:



La forma correcta de realizar un par estéreo es no tener un paralaje vertical. Si tomamos una dirección de visión para ambas cámaras paralela, obtendremos un plano proyectado sin ningún perfectamente acoplado.



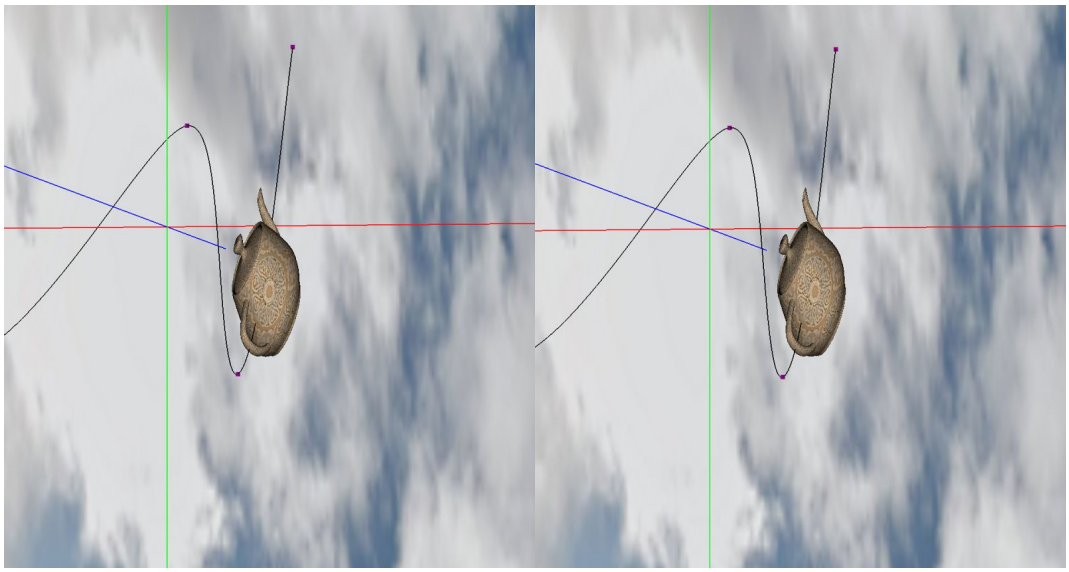
Para la implementación de este método en OpenGL se ha requerido una cámara de tipo frustrum no simétrica con los siguiente parámetros por defecto. Algunos de ellos configurables por tecla y serán referenciados más adelante:

- Campo de visión en ‘Y’: 25 grados
- Plano cercano en ‘Z’: 1
- Plano lejano en ‘Z’: 40
- Tamaño de la proyección del plano ‘Z’: 40
- Distancia intraocular: 0.2

Para que la dirección de la visión sea siempre paralela entre ambas cámaras se ha calculado el vector ortogonal a la dirección original con respecto el punto de interés para así realizar el correcto desplazamiento.

Para realizar estas funciones se ha situado los atributos necesarios en el fichero de la clase *'igvCamara'* para controlar realizar el control desde tecla.
El grueso de las funciones de la estereoscopía se encuentra en el fichero *'igvInterfaz.cpp'* en concreto en el *'set_glutDisplayFunc()' línea 228*, donde encontramos la configuración correspondiente de la cámara, de los dos viewports y su llamada a la visualización.

Captura de imagen y vídeo:



Para implementar esta funcionalidad se ha decidido por usar la librería abierta de OpenCV, la cual (entre muchas funcionalidades) nos permite la grabación en fichero de imágenes y vídeo de una forma sencilla y en un formato adecuado.
El nombre de las imágenes/vídeos generados tendrán como nombre la fecha en segundos de linux.
El formato será el genérico de *'png'* y *'avi'* para imagen y vídeo respectivamente.
Los archivos media serán guardados en la carpeta del proyecto llamada ***'Records'.***

Importante: Para activar esta función, ir al final de documento para la instalación de dependencias necesarias.

En la función *'set_glutDisplayFunc()'* del fichero *'igvInterfaz.cpp' línea 371* podemos encontrar las dos operaciones mencionadas, captura de imagen (photo) y grabación en formato de vídeo.
Para realizar esto se ha combinado ambos frameworks, tanto GLut como OpenCV. Con la orden *'glReadPixels(...)'* de OpenGL podemos obtener el valor de los pixels en la ventana actual. Estos datos los almacenamos en una variable de tipo *cv::Mat* la cual nos permitirá trabajar con ella y realizar sin mayor problemas el guardado del frame actual con *'cv::imwrite(...)'* y la captura de múltiples frames con *'VideoWriter()'*.

Fenómenos Naturales: Sistema de partículas

El elemento natural escogido para la incorporación de la escena ha sido el fuego a modo de chimenea en el modelo de la tetera.

La implementación completa de lo correspondiente a esta práctica se ha unificado en el fichero '*Particles.h',* ahí se puede encontrar la configuración global del sistema de partículas de nuestro fuego y las funciones que lo harán posible.
Este fuego consta por defecto un máximo de 800 partículas de las cuales cada una tendrá en su estructura de datos información acerca de su posición en *'X'* e *'Y'* junto con su temperatura *'tempC'*. La temperatura será un dato pseudoaleatorio que determinará tanto el tamaño de la partícula como el color asignado.
Además, nuestro sistema de partículas tendrá una altura máxima en el eje Y que determinará el punto donde morirán (*limits)* y un factor de escalado (*scaleF*).

Para la distribución del color se tomará el factor aleatorio de la temperatura para conseguir tonos intermedios a modo de fundido entre los colores bases que se han escogido para construir el fuego: Amarillo-Rojo-Negro.

Algunas condiciones y variables han sido ajustadas en las propias condiciones del código a base de prueba y error hasta conseguir la mejor sensación de

voluminosidad y realismo. Es recomendable ver el código y modificar algunas variables para mejor comprensión del mismo.

Se ha aumentado la cantidad de partículas y reducido el tamaño para tratar de conseguir un mejor efecto pero se perdía completamente la transición entre los colores por lo que se ha decidido dejar tal y como se encuentra en el proyecto y como además aparece en el vídeo de demostración.

Técnicas de producción: Cinemática Inversa

El uso que se le ha dado a la cinemática inversa ha sido para incorporar un brazo auto-dirigible a modo de aspirador o cañón. Este brazo consta de dos grados de libertad (2GL) los cuales esta técnica le permitirá calcular los dos ángulos que deberá recorrer las articulaciones para así conseguir llegar a su posición objetivo. Para mostrar esta funcionalidad se ha decidido incorporar un sistema aleatorio el cual genera un punto aleatorio en el espacio x-y de entre los posibles movimientos del brazo tras finalizar cada animación. Por cada objetivo aumentará la velocidad del movimiento del brazo hasta llegar a un máximo de velocidad, esto hará la animación más divertida.

En lo que respecta a la implementación se ha creado una función en el fichero '*igvEscena3D.cpp*' llamado '*cinematicaInversa()*' el cual se encargará de realizar el grueso de la práctica, tanto la generación del punto objetivo de nuestro brazo, el dibujado del brazo y la propia animación. El código de esta sección se encuentra lo suficiente autodocumentado como para su comprensión.
Para el cálculo de los ángulos de ambas articulaciones se han usado las siguientes fórmulas:

Ángulo del brazo:

```
float hipo = sqrt(x*x + y*y);
float alpha = atan2(y, x);
float beta = acos((longBrazo * longBrazo - longAntBrazo * longAntBrazo + hipo * hipo) / (2 * longBrazo * hipo));
angle2 = alpha + beta;
angle2 = angle2 * 180 / M_PI;
```

Ángulo del antebrazo:

```
float gamma = acos((longBrazo * longBrazo + longAntBrazo * longAntBrazo - hipo*hipo) / (2 * longBrazo * longAntBrazo));
gamma = gamma * 180 / M_PI;
angle = gamma - 180;
```

Demostración: Vídeos

DEFENSA DE PRÁCTICAS -> **https://youtu.be/GjIsoUdq3Gc**
(PR7-> https://www.youtube.com/watch?v=_eMdDJDT5og)
(PR6 -> https://youtu.be/4ALnN2gtZI8)
(PR4 -> https://www.youtube.com/watch?v=13TCRq_mgZo&nohtml5=1)
(PR3-> https://youtu.be/hAxOk1zbjBE)
(PR2 -> https://youtu.be/qV9PdWUxzLo)

- Funciones por tecla:

Tecla	Funcionalidad
'3'	Activar/Desactivar el modo estereográfico.
'.' / ':'	Aumenta/disminuye la distancia interocular en el par estéreo.
',' / ';'	Aumenta/Disminuye el grado de apertura del campo de visión en Y.
'<' / '>'	Aumenta/Disminuye la distancia del plano de proyección en Z
'l'	Intercambiar entre interpolación Lineal o Curva.
'esc'	Cerrar la aplicación.
'p'	Pausa total de la animación.
'P'	Pausa la animación del objeto.
'v'	Activar/Desactivar el escalado no uniforme.
't'	Activar/Desactivar traveling de cámara.

‘r’	Cargar los datos de fichero.
‘e’	Oculta o muestra los ejes dimensionales.
‘E’	Oculta o muestra la trayectoria de la interpolación lineal.
‘+’ / ‘-‘	Aumentar o disminuir el zoom de la cámara.
‘9’	Cambiar el modo de cámara entre Perspectiva y Paralela.

Ejecución y compilación de la aplicación:

Para la ejecución del proyecto se ha realizado una compilación estática para Windows por lo que debería ser suficiente con abrir el ejecutable **Animacion.exe** situado en la carpeta **BIN**.
En caso de no funcionar siempre se podrá compilar y ejecutar con el código fuente.

En **SOURCE** encontramos dos carpetas, una de ellas con el proyecto completo para Visual Studio 2015 y la otra con los ficheros necesarios para la construcción del proyecto (en caso de necesitarlo). Es importante recordar que la aplicación no funcionará correctamente si los ficheros de datos de entrada *“keyFrames.txt” y “traveling.txt”, “speedCurve.txt” y la carpeta de texturas* no se encuentran dentro del directorio del proyecto de Visual Studio.

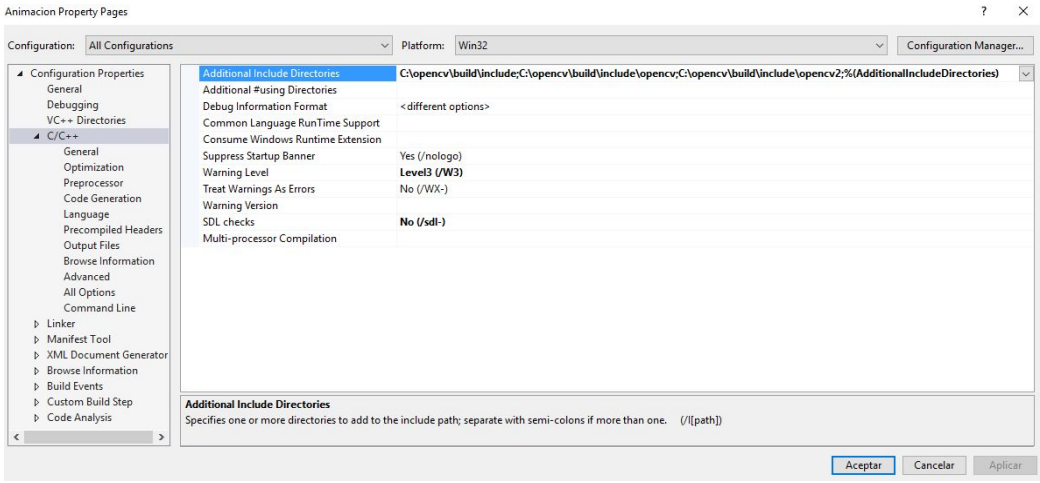
- Instalación de OpenCV: [No es necesario, solo para captura y grabación]

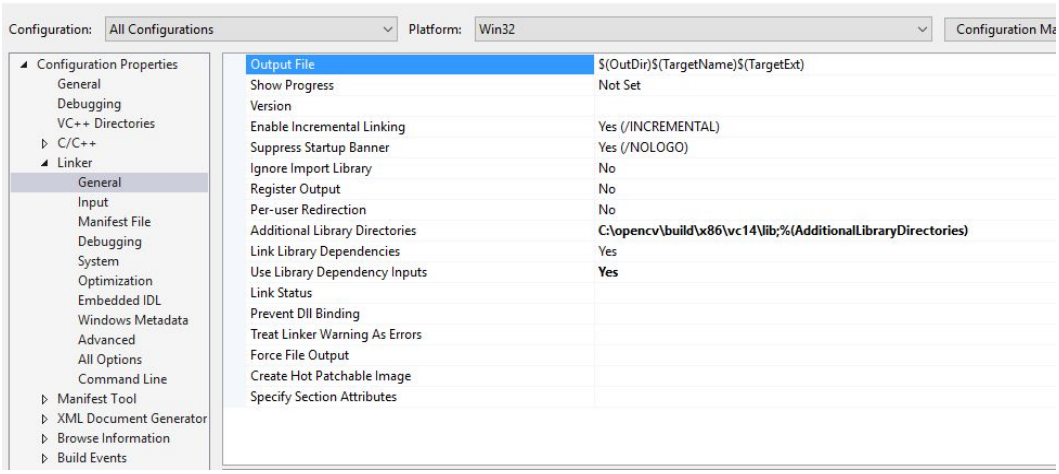
Para mayor comodidad se ha desactivado las funcionalidades de grabación de vídeo y captura de imágenes para no prescindir de las dependencias de las librerías. En caso de querer activarlo siga las siguientes instrucciones.

Para comenzar se deberá descargar y copiar a la raíz del disco **C:** esta carpeta **OPENCV**. Aquí se darán las librerías necesarias para OpenCV VC14 x86.

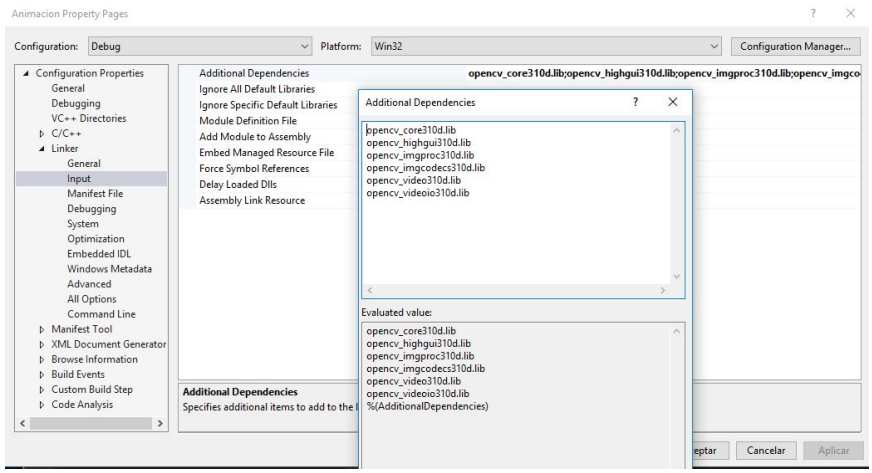
Ahora se deberá comprobar en Visual Studio 2015 las propiedades para establecer las rutas de la librería.

- Deberá quedar así:





Y el Linker -> Input -> Additional Dependencies deberemos tener al menos estas bibliotecas en la configuración de Debug:



opencv_core310d.lib
opencv_highgui310d.lib
opencv_imgproc310d.lib
opencv_imgcodecs310d.lib
opencv_video310d.lib
opencv_videoio310d.lib

Una vez hecho esto, se deberá ir a '*IgvInterfaz.cpp*' y cambiar a "true" el valor de '*#define EnableOpenCV*'