Math BackGround

* [**00:00:00**](https://youtube.com/watch?v=DnkU4_DttGE&t=0) In this section, the speaker introduces the topic of vectors and briefly touches on matrices in the context of computer graphics. They explain that vectors can be 1D, 2D, or 3D, and are frequently used in graphics algorithms. The speaker emphasizes that although vectors can have n dimensions, they will primarily focus on 2D, 3D, and 4D vectors. The preferred notation for vectors is in column format, but for readability purposes, row format may also be used by transposing the vector. The speaker notes that vectors are an important component in graphics and will be extensively used throughout the course.
* [**00:05:00**](https://youtube.com/watch?v=DnkU4_DttGE&t=300) In this section, the speaker discusses the concept of scalar values in graphics and their different meanings. Scalar values can represent either a position in three-dimensional space or a direction with length. Although they are different, they are closely related. The speaker emphasizes the importance of understanding their distinction but also their connection. They use the example of a vector with values 5, 7, and 3 to illustrate the ambiguity between position and direction. To fully understand the meaning of these values, one needs to know the units, x, y, and z directions, and the origin. Context is crucial for the interpretation of vectors, as they are meaningless without it. Overall, understanding the context is fundamental when working with graphics and coordinate frames in order to properly interpret and manipulate vectors.
* [**00:10:00**](https://youtube.com/watch?v=DnkU4_DttGE&t=600) In this section, the speaker discusses the concept of vectors in computer graphics. They explain that vectors can have two different meanings - a position in a coordinate frame and a direction from the origin. In computer graphics, a single type is typically used to represent vectors since they can be seen as interchangeable. The speaker also introduces the mathematical notation for vectors, using bold lowercase letters to denote a vector and non-bold, italicized letters for the components. They mention the importance of unit vectors, which have a length of one unit and are frequently used in computations. The speaker then demonstrates vector operations such as addition and subtraction, emphasizing that vectors require context for a meaningful interpretation.
* [**00:15:00**](https://youtube.com/watch?v=DnkU4_DttGE&t=900) In this section, the video discusses the dot product and its various uses in computer graphics. The dot product takes two vectors and forms a scalar by multiplying their x, y, and z components and adding them together. The video explains that the dot product can be used to calculate the projection of one vector onto another, as well as the angle between two vectors. If the vectors are perpendicular, the cosine of the angle between them will be zero, resulting in a dot product of zero.
* [**00:20:00**](https://youtube.com/watch?v=DnkU4_DttGE&t=1200) In this section, the speaker discusses the concept of the dot product for determining if two vectors are perpendicular to each other. They also mention some dot product identities, such as the ability to swap the order of vectors or multiply one vector with a scalar and move the scalar to the other vector. The speaker then introduces the cross product, which forms a third vector that is perpendicular to both of the original vectors. In 2D, the cross product becomes a scalar representing the area of a parallelogram, while in 3D, it becomes a vector representing the area of the parallelogram. The speaker highlights that if two vectors are in the same direction, the cross product will have a length of zero.
* [**00:25:00**](https://youtube.com/watch?v=DnkU4_DttGE&t=1500) In this section, the speaker introduces the concept of the zero vector and explains that it has no discernible direction because it is comprised of all zeros. The speaker then goes on to discuss the cross product of two vectors and explains that it can be found by flipping one vector's direction, resulting in the opposite direction. The speaker also mentions the use of a right-handed coordinate frame to determine the direction of the cross product. The speaker briefly mentions matrices and their use in computer graphics, but indicates that they will be discussed in more detail later.
* [**00:30:00**](https://youtube.com/watch?v=DnkU4_DttGE&t=1800) In this section, the speaker explains that matrices play a crucial role in computer graphics, particularly in operations like solving optimization problems and transforming vectors from one coordinate frame to another. The speaker mentions that for most 2D graphics, a 2x2 matrix is used, while for 3D graphics, a 3x3 or 4x4 matrix is used. The speaker also introduces the concept of multiplying a vector by a matrix, which is the basic operation used in graphics transformations. The speaker clarifies that the fourth dimension in matrices is not related to time but rather a concept called homogeneous coordinates. They assure the viewer that matrices are not commonly used for handling time-related stuff in computer graphics. Lastly, the speaker mentions that they will also be multiplying matrices by other matrices and highlights the important point that the order of multiplication matters. Overall, the goal of this section is to give the viewer a basic understanding of matrices and vectors as they relate to computer graphics.

**Raster Images (Part 1)**  
[**00:00:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=0) In this section, the lecturer introduces the concept of raster images and explains the difference between raster and vector images. They showcase an SVG file as an example of a vector image and explain how vectors are used to define the shapes and coordinates of the image. However, the focus of today's lecture is on raster images, which are made up of pixels. The lecturer explains that pixels are picture elements and that a raster image is formed by a collection of pixels, each with its own color value. They provide a visual example of a raster image to illustrate this concept.

* [**00:05:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=300) In this section, the speaker discusses resolution and color in raster images. They explain that resolution refers to the width and height of an image in pixels, with higher resolutions resulting in more pixels. They then move on to explain color, stating that it is a complicated topic that is often misunderstood. They clarify that color is a perceptual experience and a shared hallucination among humans, created by the combination of different wavelengths of light that our eyes perceive. The speaker also touches on cones in our eyes that perceive color and how our species agrees on what to call specific colors based on our shared perception.
* [**00:10:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=600) In this section, the speaker discusses how the perception of color is specific to human beings and varies among different animals. Humans have three different types of cones that are sensitive to different wavelengths of light, resulting in the perception of blue, green, and reddish colors. These three values form the basis of the RGB (red, green, blue) color model used in computer graphics. By combining different intensities of these three colors, it is possible to generate a wide range of colors that humans can perceive. The speaker also mentions that color values are typically represented as a range between 0 and 1, with 0 representing no light and 1 representing maximum light intensity. This RGB color model can be visualized as a cube with red, green, and blue forming the three coordinate directions.
* [**00:15:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=900) In this section, the speaker explains the concept of color space and dynamic range in raster images. They introduce the RGB color space and discuss how white represents the maximum light that typical monitors can generate. They also mention the existence of high dynamic range displays that can generate brighter light. The speaker explains that low dynamic range displays, which are commonly used today, have a range between zero and one. They then discuss the quantization of color values, where 8 bits are commonly used to represent the intensity values for each color channel. However, they mention that if images are to be modified, using more precision, such as 16 bits, may be necessary to avoid losing information due to quantization.
* [**00:20:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=1200) In this section, the speaker discusses the different ways of storing color information in raster images. The two general methods are interleaved format, where each pixel's RGB color is stored consecutively, and separate channels, where each color channel is stored independently. Most commonly, images are stored in an interleaved format, as it allows for better compression and is easier to work with. The speaker also mentions that for low dynamic range images, 8 bits per channel is sufficient, while for high dynamic range images, 32-bit floating point values are used during computations and then converted to corresponding low dynamic range values. Some image file formats use 16-bit floating point values for storage to save space.
* [**00:25:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=1500) In this section, the speaker discusses raster image formats, starting with the bitmap and portable text map image files (BMP and PPM). These formats are simple and store uncompressed images in RGB values. The speaker also mentions that these formats are not very popular compared to other raster image formats. They then introduce the PNG format, which is a modern graphics format known for its lossless compression. PNG files can be used for storing and sending files, but they are uncompressed and displayed in their original format. The speaker explains that PNG files can store 8 or 16 bits per channel, or they can use a color table to compress images with fewer colors more efficiently. However, using a color table introduces some loss in image quality. Overall, PNG files are primarily designed for storing LDR (low dynamic range) images.
* [**00:30:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=1800) In this section, the video discusses the use of color tables in raster image file formats. The speaker explains that PNG and GIF file formats use color tables, which can contain 2 to 256 colors specifically chosen for each image. The compression of PNG images comes from both the color table and a lossless compression scheme. On the other hand, JPEG, a popular file format for photographs, uses lossy compression, meaning that the compressed image loses some information and cannot exactly reproduce the original image. However, JPEG compression allows for significant reduction in file size, with typical JPEG images using eight bits per color. The video then demonstrates the difference in file sizes between uncompressed BMP, compressed PNG, and low-quality JPEG files using a low-resolution image, highlighting the significant reduction achieved by PNG compression. Finally, the video shows the poor quality of a low-quality JPEG image due to the loss of information during compression.
* [**00:35:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=2100) In this section, the speaker explains how raster images store differences from the average color, allowing for compression with very few bits. They demonstrate this compression using JPEG and show that the quality of the image can be adjusted based on the number of bits used. They also compare JPEG compression to GIF compression, showing that GIF compression with color quantization can achieve similar results to JPEG without losing image quality. However, they acknowledge that JPEG compression may not always perform well and show examples where PNG compression is more effective. Overall, they highlight the trade-offs between image quality and storage size when using different compression methods.
* [**00:40:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=2400) In this section, the speaker discusses the differences between various image compression formats and the importance of choosing the right format based on the type of image. They show examples of JPEG compression with different quality settings and compare it to PNG and GIF formats. The speaker explains how GIF compression can result in color quantization artifacts due to limited color options, while JPEG compression can provide better quality with similar file sizes. They also mention how image conversion software considers neighboring pixels and error accumulation during quantization to minimize the average error and create a visually pleasing image. Overall, the speaker emphasizes the necessity of understanding the image data and selecting the appropriate file format to achieve the desired result.
* [**00:45:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=2700) In this section of the video, the speaker discusses the compression artifacts and color values in raster images, particularly JPEG format, which does a better job at preserving color values. They also mention that GIF format is good for animations and that PNG format also supports animation, but is not widely supported by all browsers. The speaker then moves on to HDR (high dynamic range) file formats, explaining that they are relatively new and designed for storing full-color information without clipping. They discuss the specific HDR format called EXR, which is used in film production due to its ability to store high-quality images without losing any information. The speaker also mentions that HDR images are more expensive due to the larger file sizes and less effective compression algorithms compared to regular 8-bit images. However, they are highly valued in computer graphics and film production for their ability to preserve all the details in the image.
* [**00:50:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=3000) In this section, the speaker discusses the storage and conversion of raster images. They mention that storage is not expensive nowadays, so it's acceptable to save images in an uncompressed format. When images are read from the disk or received via the internet, they are converted into an uncompressed string of RGB values before being sent to the display. The standard way of storing images is in the scanline order, but this may not be the most efficient for all algorithms. In certain cases, it makes more sense to store images as a 2D construct instead of a 1D scanline to improve cache efficiency. GPUs often use swizzle orders for storing images, such as the z-curve order shown. Different compression algorithms are used for real-time image manipulation on GPUs compared to those used for storing images to disk.
* [**00:55:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=3300) In this section, the speaker briefly mentions image compression on GPUs, explaining that GPUs can compress images losslessly in order to benefit from compression without the user being aware. They also talk about the differences between lossless and lossy compression, with lossy compression requiring more careful consideration. In response to a question, they mention that image compression algorithms are not sensitive to the specific colors being stored, but they can be sensitive to changes in color values. They also explain that computer screens displaying black may still emit some light due to backlighting and display technologies. Additionally, they mention that there are formats that allocate different amounts of bits to different color channels, with green often being allocated more bits due to its importance in light sensitivity. Some displays also manipulate images by using higher bit representations, even if the input is lower bit.
* [**01:00:00**](https://youtube.com/watch?v=zllIPDaiOyk&t=3600) In this section of the video, the speaker discusses the different formats and variations in the number of bits used in raster images. They also address a question about the difference between HDR (high dynamic range) and LDR (low dynamic range) storage. HDR images allow for a wider range of colors and intensities, preserving data that LDR images may clip. Additionally, HDR images store more information in between color values, although the speaker notes that using 16 bits per color channel is generally sufficient for most purposes.

**Intro to Graphics 04 - Raster Images (Part2)**

* [**00:00:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=0) In this section, the lecturer continues the discussion on raster images and begins by explaining the concept of gamma in computer graphics. Gamma refers to the relationship between light intensity and the values used to represent it in digital images. The lecturer uses the example of a grayscale image to demonstrate that humans are better at differentiating between darker shades of gray than lighter ones. This means that when mapping a range of light intensities onto a limited number of binary values, it is more efficient to allocate more values to the darker end of the spectrum. This ensures that there is enough precision to distinguish between similar shades of dark gray, while still providing enough representation for lighter shades.
* [**00:05:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=300) In this section, the speaker discusses the concept of gamma correction in displays. They explain that displays allocate more discrete values to darker intensities and fewer values to brighter colors, which aligns better with the human visual system. This allocation is achieved by using a gamma function, which takes the input color or intensity value and raises it to the power of gamma. The resulting output intensity is then scaled by the gamma value to ensure the desired intensity is achieved on the display. The typical gamma value used is 2.2, and this gamma correction ensures that images displayed on monitors are accurately represented.
* [**00:10:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=600) In this section, the speaker discusses the concept of gamma correction in relation to image file formats and display devices. They explain that gamma correction is necessary because different monitors use different gamma values, but the standard today is around 2.2, which is related to the sRGB standard. The speaker also mentions that sRGB uses a more complicated function than a simple exponent function, but for most graphics applications, using a gamma value of 2.2 works well. They further explain the use of LEDs in displays, where each pixel typically has three LEDs (one for each color), and how cameras differ in their function of sensing light instead of producing it.
* [**00:15:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=900) In this section, the speaker explains how digital cameras capture colors using a pattern of sensors, rather than capturing all RGB color components for each pixel. They use a Bayer pattern, where each pixel captures only one of the color values. The green component is captured more accurately than red and blue because our visual range is more sensitive to green intensities. The other color values are interpolated using neighboring values. This leads to higher-quality detail in green and lower resolution in red and blue. The speaker also mentions that cheaper cameras may have 1D arrays for sensors instead of 2D arrays, which can result in shifting and skewing effects in images. The proper way to think about a pixel is as a color value defined at the center of its location, rather than as a little square.
* [**00:20:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=1200) In this section, the speaker discusses pixels and how they are used in computer graphics. They explain that pixels are displayed as little squares on display devices and that they represent colors at the center of the pixel. The speaker also mentions that images with high frequencies, such as sharp edges, may not be properly represented with low-resolution images. To improve the image quality, they suggest blurring out the sharp edges. The concept of alpha blending is introduced as a way to combine multiple images using transparency information. Alpha values are used to indicate the opacity of a pixel, with 0 being fully transparent and 1 being fully opaque. Each pixel is represented using RGBA values, with 8 bits per channel, resulting in a 32-bit pixel value. The speaker also mentions that images are typically stored in an interleaved format, with each pixel having its own RGBA values.
* [**00:25:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=1500) In this section, the speaker discusses alpha blending in raster images. They explain how to combine the color and alpha values of a foreground image with a background image using a linear interpolation formula. The formula takes into account the opacity of the foreground image to determine how much of the foreground color to show and how much of the background color to show. The speaker also addresses the scenario where the background image has an alpha value and explains how to properly combine the alpha values of both images. Overall, the concept of alpha blending allows for the composition of images with varying levels of transparency.
* [**00:30:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=1800) In this section, the speaker discusses the concept of foreground and background alpha values in relation to color blending. They mention that if the background alpha value is one, then the resulting equation simplifies to a more straightforward form. However, if the resulting alpha value is zero, it can only happen if both the foreground and background alpha values are zero. In this case, the speaker explains that it doesn't matter what color is chosen, as it will disappear anyway. They also introduce the concept of pre-multiplied alpha and straight alpha, stating that most images are recorded in the straight alpha form, but when displaying them, the color values need to be multiplied by their alpha values. The speaker suggests that it makes sense to store the values as pre-multiplied alpha to save computation.
* [**00:35:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=2100) In this section, the speaker discusses the concept of pre-multiplied alpha in raster images. When using pre-multiplied alpha, the color information gets destroyed when multiplying it by zero. Even with small alpha values, there is a loss of precision due to quantization. Storing images using straight alpha instead of pre-multiplied alpha is more common because it doesn't result in information loss. The speaker also mentions that alpha blending with a background image produces a pre-multiplied alpha color unless divided by the alpha value. There are different blending modes available, such as additive blending, which simply adds the intensity values of the foreground and background images. GPUs typically support this blending function and it can be useful in certain cases. However, when working with images that have alpha channels, the blending process becomes slightly more complex. The alpha values need to be clamped within the valid range of 0 to 1, and the resulting color values need to be modified using alpha.
* [**00:40:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=2400) In this section, the speaker discusses different blending modes and their effects on raster images. They start by explaining the additive blending mode, which involves adding the color values of the foreground image on top of the background image. They mention that the final result cannot be brighter than white, and demonstrate how this blending mode can overexpose certain pixels. The speaker then introduces the difference blending mode, which calculates the absolute difference between two colors. This mode is commonly used for comparisons in computer graphics. They also discuss the multiply blending mode, which involves multiplying the colors of the foreground and background images. This mode is useful for adding dark elements to an image without completely blackening it. The speaker explains how to create the illusion of an alpha channel for images that don't have one. Finally, they mention the screen blending mode, which is the opposite of multiply blending and results in brighter colors overpowering darker ones.
* [**00:45:00**](https://youtube.com/watch?v=PMIPC78Ybts&t=2700) In this section, the speaker discusses different blending modes in graphics, specifically focusing on screen blending. They explain that to achieve screen blending, one takes the inverse of the first image and multiplies it by the inverse of the second image. The resulting image is then inverted again to obtain the final screen value. The speaker mentions that screen blending is a useful tool and is often used in practice. They also highlight that there are many other blending modes available in Photoshop, such as multiply and various light modes. The speaker concludes by emphasizing the importance of alpha blending as a standard way of blending two images.

**2D Transformation**

* [**00:00:00**](https://youtube.com/watch?v=EKN7dTJ4ep8&t=0) In this section, the speaker introduces the topic of 2D transformations in computer graphics. They explain that transformations involve moving and manipulating objects within a scene, just like in reality. These transformations are done mathematically using vectors and matrices. The speaker focuses specifically on affine transformations, which include translation, rotation, and scaling. They mention that skew is often included in affine transformations but explain that it is actually a combination of rotation and scale. The speaker emphasizes the importance of affine transformations in computer graphics as they preserve lines and parallel lines. They mention that affine transformations are used frequently in building scenes and rendering in computer graphics. The speaker also mentions that they will discuss extending these transformations to 3D in the next section.
* [**00:05:00**](https://youtube.com/watch?v=EKN7dTJ4ep8&t=300) In this section, the speaker introduces the concept of 2D transformations in computer graphics. They start with translation, which involves moving a vector from one position to another by adding a translation vector to it. The speaker then explains scale, which allows for making objects larger or smaller by multiplying the vector components by a scalar factor. They differentiate between uniform scale, where both x and y components are scaled equally, and non-uniform scale, where the scaling amounts differ for each component. Finally, the speaker explores rotation, where the vector is rotated by an angle theta. They break down the vector into its x and y components and demonstrate how both components are rotated together. Overall, the speaker emphasizes that these transformations involve simple mathematical operations.
* [**00:10:00**](https://youtube.com/watch?v=EKN7dTJ4ep8&t=600) In this section, the instructor explains how to find the coordinates of a rotated vector using trigonometric identities. The x coordinate of the rotated vector is determined by multiplying the magnitude of the original vector by the cosine of the rotation angle, while the y coordinate is determined by multiplying the magnitude by the sine of the angle. The instructor also introduces the concept of rotation matrices, which are used to simplify the notation for rotations. These matrices are orthogonal, meaning that their columns form perpendicular vectors. The instructor mentions that the notation for clockwise rotations is slightly different, with the rotation angle being negative.
* [**00:15:00**](https://youtube.com/watch?v=EKN7dTJ4ep8&t=900) In this section, the speaker discusses the concept of orthogonal matrices and their relationship to rotation and reflection. They explain that when a vector is rotated, the resulting rotated components will still be perpendicular to each other. This concept is related to the fact that rotation matrices are orthogonal matrices. Furthermore, the speaker reveals an important property of orthogonal matrices - their inverse is equal to their transpose. They demonstrate how this property holds true for rotation matrices. However, they caution that not all orthogonal matrices are rotation matrices, as reflections can also be included. The speaker then moves on to the topic of scaling and introduces the idea of representing non-uniform scaling as a matrix multiplication, making it easier to perform scale operations. These scale matrices are simple diagonal matrices, where non-zero values are along the diagonal.
* [**00:20:00**](https://youtube.com/watch?v=EKN7dTJ4ep8&t=1200) In this section, the speaker discusses the representation of scale as a matrix in 2D transformations. They explain that any two by two matrix can be represented as a combination of rotation, scale, and another rotation. They introduce the concept of singular value decomposition, which separates a matrix into three matrices: two orthogonal matrices (representing rotations) and a diagonal matrix (representing scale). They emphasize that the important aspect is the ability to convert any series of rotation and scale operations into a single transformation matrix. However, they acknowledge that an important component, translation, is missing from this representation and discuss a possible solution for incorporating it.
* [**00:25:00**](https://youtube.com/watch?v=EKN7dTJ4ep8&t=1500) In this section, the speaker discusses the concept of combining rotations and translations in graphics. They explain that if you want to rotate an object around its center, you would need to first apply a translation to move the center to the origin, then apply the rotation, and finally translate it back to its original position. However, this process can become messy when multiple rotations and translations are involved. To address this, the speaker introduces the concept of homogeneous coordinates, which involves adding an extra coordinate to vectors and using a matrix to represent translations. By using homogeneous coordinates, the speaker aims to simplify the process of combining rotations and translations.
* [**00:30:00**](https://youtube.com/watch?v=EKN7dTJ4ep8&t=1800) In this section, the speaker discusses the concept of homogeneous coordinates and how they can be used to represent transformations in computer graphics. By using matrices, multiple affine transformations (translation, rotation, scale) can be combined into one matrix, which represents the result of the entire transformation. The resulting matrix consists of a bottom row with zeros and a one, representing translation, and a 2x2 matrix representing rotation and scale. This single matrix is a powerful tool in computer graphics as it allows for the easy manipulation of objects through various transformations. The speaker also explains that transformations can be thought of as either changing a vector in a coordinate frame or transforming the coordinate frame itself.
* [**00:35:00**](https://youtube.com/watch?v=EKN7dTJ4ep8&t=2100) In this section, the speaker discusses the concept of vectors representing positions and directions, and how they can be transformed using the same matrix. They explain that vectors can be thought of as either representing a deformed object or a viewpoint from a different coordinate frame. The speaker also introduces the idea of homogeneous coordinates, which allow for easily handling vectors that represent directions without affecting their translation component. This allows for the transformation of both position vectors and direction vectors using the same matrix.
* [**00:40:00**](https://youtube.com/watch?v=EKN7dTJ4ep8&t=2400) In this section, the speaker discusses the concept of direction vectors and how they are used to represent transformations in 2D graphics. The goal is to bring everything into a homogeneous form that simplifies the representation of transformations. However, the speaker notes that they will not be covering transformations in 3D in this video and will save that topic for a future discussion.

**3D Transformations**

* [**00:00:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=0) In this section, the lecturer introduces 3D affine transformations and explains that they can be represented using a 4x4 matrix in 3D, with the last row remaining unchanged. He then goes on to discuss the three basic transformations: scale, translation, and rotation. The rotation in 3D is more complex, as objects can be rotated around the x, y, or z-axis, or any arbitrary axis. However, he explains that any rotation around a given axis can be decomposed into rotations around the x, y, and z axes. While the process of constructing the matrix for these rotations involves some math, the lecturer will illustrate how it works without going into the mathematical details.
* [**00:05:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=300) In this section, the instructor explains that any rotation around a random axis can be decomposed into three smaller rotations. These smaller rotations can be performed around the x-axis, the y-axis, and the z-axis. The instructor provides the matrix representation for each of these rotations and explains that the axis being rotated around remains unchanged in each case. They also mention that the order of the rotations matters and can result in different outcomes. Lastly, the instructor states that any rotation can be decomposed into one of six possible combinations of these smaller rotations.
* [**00:10:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=600) In this section, the instructor explains the concept of transformation in 3D graphics and how it's used in the rendering process. He emphasizes the importance of understanding multiplication order and reminds students to apply transformations in the correct sequence. The instructor then introduces the idea of projecting the 3D scene onto a display to create a 2D image. He discusses the need to define a coordinate frame for the environment and suggests placing the origin at the center of the display. The x-coordinate is defined as left to right, while the y-coordinate is defined as up and down.
* [**00:15:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=900) In this section, the speaker discusses the concept of coordinate frames and the use of transformations in building a 3D scene. They explain that when working with raster images, the y coordinate is often flipped to align with the top left corner, while x represents left and y represents up. The z coordinate is negative behind the screen, defining the depth of the scene. They introduce the concept of camera space or view space, where x is right, y is up, and z points behind. The speaker also mentions the v volume, which represents the visible part of the scene. They explain that objects in the scene are transformed using rotation, translation, and scaling, and discuss the use of transformation matrices for this purpose. Overall, this section provides an introduction to the coordinate systems and transformations involved in creating 3D graphics.
* [**00:20:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=1200) In this section, the speaker explains how 3D transformations are used to move, rotate, and scale objects in software. They mention that these transformations are represented by a transformation matrix, which is computed and applied to objects. The speaker also discusses the concept of viewing transformations, where the scene is converted into a space similar to the view space. They explain the process of positioning and orienting a camera in the scene to define the view transformation. The camera's position, orientation, and up vector are used to construct the transformation matrix. The speaker briefly mentions that the transformation can also be constructed through translation, rotation, and scale. Ultimately, the scene is transformed into camera space, preparing it for projections.
* [**00:25:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=1500) In this section, the instructor discusses how to define the view volume for projection transformations in 3D graphics. The view volume is defined by the minimum and maximum values of x, y, and z coordinates, which determine the boundaries of what will be visible on the screen. The instructor explains that these boundaries can be converted into a standardized volume called the canonical view volume, which is a unit cube centered at the origin. The goal is to transform the camera space positions into the canonical view volume through a process called projection transformation. The instructor also mentions that while the image will be formed on one side of the canonical view volume, the 3D nature of the data is preserved for other operations.
* [**00:30:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=1800) In this section, the speaker introduces the concept of orthographic projection in 3D transformations. They explain that orthographic projection involves scaling, translating, and converting the camera view space into the canonical view volume. The limits of the camera space are defined by left and right values for x, bottom and top values for y, and near and far values for z. The speaker then discusses the transformation matrix needed to convert from camera space to canonical view space, noting that there is no rotation involved, only non-uniform scaling and translation. They provide the matrix representation for this transformation and explain how it works by normalizing the x, y, and z values. Overall, this section provides an overview of how orthographic projection is achieved in 3D graphics.
* [**00:35:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=2100) In this section, the speaker explains the process of achieving an orthographic projection by applying scale and translation transformations. They demonstrate how the x-coordinate is multiplied by the given x-value, while the y and z coordinates are ignored. The speaker also highlights that orthographic projections lack perspective, resulting in a design document-like appearance rather than a realistic image. However, they note that this projection has the advantage of preserving object scale regardless of their distance from the camera. To contrast, the speaker briefly shows an example of a perspective projection, which shows the expected diminishing size of objects as they move further away.
* [**00:40:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=2400) In this section, the speaker introduces the concept of perspective projection and explains why it is used in rendering. They mention that our eyes and cameras have a natural perspective view, unlike an orthographic projection. The shape that needs to be transformed for perspective projection is described as a pyramid-like volume. The goal is to deform this shape into a form that resembles the desired perspective projection. The speaker then discusses the idea of transforming lines originating from the origin into parallel lines to achieve this. They emphasize the need to understand the process rather than simply memorizing formulas for perspective transformation.
* [**00:45:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=2700) In this section, the speaker discusses how to convert a line segment in the y-z plane to a line segment in the z direction using perspective transformation. They explain that the y-value remains constant for the entire volume, and therefore, the ratio of y divided by z is the same for all points along that line. By multiplying this ratio with the z-value and a scaling factor, the speaker is able to transform the line segment into the desired line in the z direction. They also note that this same concept applies to the x-axis as well.
* [**00:50:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=3000) In this section, the speaker discusses the concept of homogeneous coordinates and how they can be used to represent 3D points in a 4D space. By extending the definition of homogeneous coordinates, the speaker explains that all points with the same fourth coordinate are equivalent and can represent the same 3D position. This concept allows for transformations of 3D points using matrix multiplication. The speaker mentions that they will be manipulating 3D positions using a 4D vector, where the x and y coordinates are multiplied by a constant and the z coordinate is represented as a question mark. They explain that representing the vector in this form makes the division calculation easier. Overall, this section introduces the concept of homogeneous coordinates and their significance in 3D transformations.
* [**00:55:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=3300) In this section, the speaker discusses perspective transformation and how to represent it using a matrix multiplication. They explain that the x and y coordinates can be scaled by a factor of n, and the fourth coordinate is determined by the z value. However, they encounter a challenge with the z component and explain their solution of keeping the z values unchanged while minimizing modification. They introduce the near and far values and demonstrate how the matrix multiplication represents the output z value based on the input z value. Overall, the speaker explains the process of perspective transformation and the considerations involved.
* [**01:00:00**](https://youtube.com/watch?v=1z1S2kQKXDs&t=3600) In this section, the narrator discusses the use of perspective transformation and orthographic projection in 3D graphics. They explain that perspective transformation involves modifying the z coordinate while keeping the extent of the z coordinates the same. They also mention that the order in which the transformations are applied is important, with the perspective transformation matrix coming before the orthographic projection matrix. The narrator highlights the importance of understanding these concepts for graphics operations and mentions that while the upcoming projects will focus on 2D transformations, 3D transformations will be used in future projects.

**GPU Pipeline**

* [**00:00:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=0) In this section, the lecturer introduces the GPU pipeline and its importance in hardware-accelerated rendering. They clarify that when they refer to the GPU, they are specifically referring to the graphics processing unit, not the entire video card. The lecturer emphasizes that they will focus on a high-level understanding of the GPU's function and its operation from the perspective of the graphics API. They explain that while the GPU itself has undergone significant changes over the years, the graphics API has remained relatively stable. The primary function of the GPU is to produce images, and its role as an image-making machine is foundational.
* [**00:05:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=300) In this section, the speaker discusses the concept of what a GPU does in terms of compute graphics. The GPU takes in scene data, which is a collection of triangles, and other information that defines the image to be produced. This scene data needs to be stored in GPU memory before any processing can be done. The speaker emphasizes the importance of transferring the data from CPU memory to GPU memory. Once the data is in GPU memory, it can be processed by the GPU pipeline, which involves applying transformations to the scene data and rasterization, ultimately producing the final image.
* [**00:10:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=600) In this section, the speaker explains the GPU pipeline, which involves converting vector data into raster image data. The vector part of the pipeline performs various operations on the scene data to transform it into a canonical view volume. The rasterizer is a hardware unit that generates fragments (pixels) from the primitives, such as triangles. The fragments are then colored, and alpha blending takes place to form the final image. The pipeline consists of three components: the vertex shader, which handles transformation; the rasterizer, which performs rasterization; and the fragment shader, which computes the color of each fragment. The vertex shader and fragment shader are programmable units, while the rasterizer is implemented in hardware. The speaker emphasizes that the main control lies in the vertex shader and fragment shader, with additional hardware operations before and after them.
* [**00:15:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=900) In this section, the speaker explains the GPU pipeline and its various stages. They mention that a whole bunch of triangles, each with three vertices, are sent to the vertex shading stage, where each vertex is transformed into canonical view space. Then, there is a hardware unit that takes these transformed vertices and forms the triangles before giving them to the rasterizer. They also mention optional shader stages such as tessellation and geometry shaders, which can split primitives into smaller triangles and quads or perform custom operations before the rasterizer. However, in WebGL, only the required vertex shader and fragment shader are accessible. The speaker mentions that there is more complexity in the pipeline, including newer concepts like mesh shaders, but for the purpose of this course, they focus on the simplified picture of the required part of the rendering pipeline. They explain that the CPU application controls all these stages, and in this course, they use JavaScript to control the GPU from the CPU application.
* [**00:20:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=1200) In this section, the speaker explains how data is communicated between the CPU and GPU in a graphics application. The CPU application sends data, such as scene data, from the CPU memory to the GPU memory. Once the rendering pipeline is ready to run, the GPU starts consuming data from its memory. If there is dedicated hardware to run WebGL, the CPU will simulate the pipeline. It is important to note that even on devices with integrated GPUs, there is still a distinction between CPU and GPU memory. The speaker also mentions that GPU code, such as that for vertex and fragment shaders, is not pre-compiled but is compiled by the graphics driver during runtime. Similarly, JavaScript code in the application is not compiled either.
* [**00:25:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=1500) In this section, the speaker discusses the compilation process for GPU code and the role of the graphics API in setting parameters for vertex and fragment shaders. The CPU code sets the parameters for the shaders based on certain criteria, such as user inputs or knobs, and communicates with the graphics API to send the code and parameters to the graphics driver, which then communicates with the GPU hardware. The layered cake model is used to visualize this process, but the speaker prefers a different visualization where all arrows originate from the graphics API. Different APIs are available for different programming languages, such as OpenGL for C++ or DirectX for Windows devices, and Apple has its own API called Metal for Apple devices.
* [**00:30:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=1800) In this section, the speaker introduces the WebGL API as the graphics API they will be using throughout the course. They explain that WebGL is similar to OpenGL but is a simplified version that is designed for web applications and works on most modern browsers. They highlight that WebGL allows developers to access the capabilities of the GPU within a web page, making it a powerful tool for learning about graphics programming. The speaker also mentions how the canvas element in HTML serves as a container to display the images rendered by the GPU. They provide code examples and explain the process of initializing WebGL and obtaining the WebGL context.
* [**00:35:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=2100) In this section, the speaker discusses the process of creating a WebGL context in JavaScript and explains that the context is like a class that contains various functions used to control the GPU. The context is responsible for connecting the CPU code to the GPU driver and contains all the necessary information and data for the application to access the GPU. The speaker also mentions that additional parameters can be set when creating the context, and that the width and height properties of the canvas determine the output image resolution. The viewport is also set to cover the entire canvas, allowing for rendering of multiple image pieces if needed.
* [**00:40:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=2400) In this section, the speaker explains the concept of device pixel ratios and how they affect the scaling of web pages. They demonstrate adding the pixel aspect ratio to their JavaScript code in order to determine the actual pixel size on the display. This allows them to render images at the native resolution of the display, rather than relying on the operating system to scale them. The speaker also covers other initializations, such as setting the clear color for the screen and initializing line widths. They mention that these initializations should be done after the web page is loaded to ensure proper execution.
* [**00:45:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=2700) In this section, the video discusses the process of setting up the scene data in WebGL. The scene data consists of primitives, such as points, lines, and triangles, which are defined by a set of vertices. Each vertex can have attributes like position, color, and other customizable properties. The scene data is then passed to the GPU, which sends it to the vertex shader. The vertex shader code, written by the developer, has the flexibility to interpret and manipulate the data as needed. This allows for complete customization and control over the scene data in WebGL.
* [**00:50:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=3000) In this section, the speaker discusses the process of sending data to the GPU for rendering. They explain that the data, such as vertex positions and colors, need to be converted into arrays that can be understood by the GPU. The winding order of vertices and the order in which they are provided is discussed, with the speaker mentioning that it can be important depending on the rendering settings. They also mention the concept of allocating memory on the GPU and creating buffers to store the data. The binding of the buffers is also highlighted as an important step in the OpenGL handling process.
* [**00:55:00**](https://youtube.com/watch?v=UzlnprHSbUw&t=3300) In this section, the speaker explains the process of binding and sending data to the GPU memory. They first bind a buffer and then send the corresponding data to that buffer, without explicitly specifying the size of the array. The speaker demonstrates this with position and color data, creating buffers for each and pushing the data into the GPU memory. They also note that only one buffer can be bound at a time. The speaker points out that memory allocation on the GPU doesn't happen until the data is sent, and this is where the size of the data is determined. Overall, the speaker concludes this section by stating that they have finished putting the scene data in the GPU memory and will move on to discussing shader programs and sending data in the next session.

**WebGL**

* [**00:00:00**](https://youtube.com/watch?v=je_PhVKJYng&t=0) In this section, the lecturer introduces the concept of the GPU pipeline, which consists of the CPU application, the graphics API (in this case, WebGL), and the GPU. The lecturer explains that the CPU application uses the graphics API to control the GPU for rendering purposes. The pipeline involves setting up the scene, creating buffers for vertices, writing and setting parameters for shaders, and initiating the rendering process. The GPU takes all this data and produces an image. The lecturer also explains the process of setting up WebGL in JavaScript, which includes creating a canvas, getting the WebGL context, setting the output resolution, and setting the viewport to cover the entire canvas.
* [**00:05:00**](https://youtube.com/watch?v=je_PhVKJYng&t=300) In this section, the speaker explains the concept of setting states in WebGL and OpenGL, comparing the graphics APIs to state machines. By setting certain states, such as background color and line thickness, the GPU will use those states when rendering. The speaker demonstrates how setting the background color to white does not actually create a white image, but rather sets the state for when the screen is cleared. Similarly, the line thickness state is set for future line drawings. These states can be modified as needed, but for values that remain constant, it is common to set them once after generating the canvas and not worry about them again. The speaker also mentions the possibility of adjusting the resolution based on window size changes, which might require setting the width and height states multiple times. Additionally, the speaker notes that WebGL calls can only be made after creating the canvas and obtaining the context, emphasizing the importance of this sequence.
* [**00:10:00**](https://youtube.com/watch?v=je_PhVKJYng&t=600) In this section, the speaker explains the process of generating a WebGL canvas and setting the output image resolution. They then discuss the different types of primitives supported by WebGL, such as points, lines, and triangles, and how they can be used to define a scene. The speaker also introduces the concept of vertex attributes, such as positions and colors, and explains how to allocate memory on the GPU and transfer data from CPU to GPU. They mention the use of two separate function calls, even though it may seem more efficient to combine them, and provide reasons for this design choice.
* [**00:15:00**](https://youtube.com/watch?v=je_PhVKJYng&t=900) In this section, the speaker discusses how to modify different parts of a buffer in WebGL. They explain that a separate function called "modify" can be used to modify specific parts of the buffer instead of specifying the same buffer repeatedly. They also address a question about whether WebGL uses event listeners for things like resizing. The speaker clarifies that WebGL is strictly a graphics API and does not handle event listeners or user inputs. Instead, JavaScript is responsible for handling those aspects, as it runs on the browser and can easily interact with HTML, CSS, and handle user input. The speaker then explains the process of setting position and color data in GPU memory, but emphasizes that WebGL has no knowledge of what the data represents until explicitly told. Moving on, the speaker introduces vertex and fragment shaders, explaining that the vertex shader receives the scene data, applies transformations, and passes it to the rasterizer, which generates pixels. The fragment shader then computes the colors of the fragments. They show an example of a vertex shader written in GLSL, the shading language used for GPU shaders in WebGL and OpenGL.
* [**00:20:00**](https://youtube.com/watch?v=je_PhVKJYng&t=1200) In this section, the speaker introduces the concept of writing WebGL code separately from JavaScript code. The video demonstrates that WebGL compiles GLSL code and runs it once for each vertex. The code shown takes position and color attributes and converts the position into a four-dimensional vector by adding one to it. The resulting position is then set as the output by assigning it to the "gl\_Position" variable. The speaker also mentions that to ensure the code works correctly, the positions received as attributes should be in the canonical view volume. The speaker suggests that transformations may be needed to convert positions from a different space, such as object or world space, into the canonical view volume.
* [**00:25:00**](https://youtube.com/watch?v=je_PhVKJYng&t=1500) In this section, the speaker discusses the use of a matrix for transforming vertex positions in WebGL. They explain that the elements within "back three" and "back four" will always be floats, but GLSL also supports vectors of integers. The speaker then introduces the concept of attributes and uniforms in a shader. Attributes are used to transform vertices, while uniforms are parameters that are uniformly applied to all vertices. The speaker explains that uniforms will be set in a later section. The speaker also mentions a color attribute that is not currently used, but will be passed to the fragment shader using a varying variable. They provide the complete code for the vertex shader and the fragment shader.
* [**00:30:00**](https://youtube.com/watch?v=je_PhVKJYng&t=1800) In this section of the video, the speaker discusses the concept of vertex and fragment shaders in WebGL. The vertex shader is responsible for setting the colors of the vertices, while the fragment shader determines the output color of each pixel. The varying vector is used to pass the output from the vertex shader to the fragment shader. The GPU automatically interpolates the color values for each pixel based on the vertices, resulting in a smooth transition of colors across the triangle. The combination of a vertex and fragment shader forms a WebGL program, which is necessary for rendering on the GPU.
* [**00:35:00**](https://youtube.com/watch?v=je_PhVKJYng&t=2100) In this section, the speaker explains the different types of shaders that can be used in WebGL, including vertex shaders and fragment shaders. They mention that OpenGL requires a vertex shader and fragment shader, while WebGL only requires these two types. The process of compiling shader code is also discussed, with the speaker showing an example of how to compile a vertex shader using WebGL commands. They emphasize the importance of checking the compile status to ensure that the code was compiled successfully. If there are compilation errors, the speaker advises fixing the code and not proceeding until the errors are resolved. They stress the need to include error messages or alerts in the code to easily identify any JavaScript or GLSL compilation errors.
* [**00:40:00**](https://youtube.com/watch?v=je_PhVKJYng&t=2400) In this section, the speaker explains how to compile and link the vertex and fragment shaders in WebGL. The process involves creating the shaders, compiling them, and then linking them to the shader program. The speaker also emphasizes the importance of checking for linker errors after linking the shaders. They also mention that the advantage of compiling shaders at runtime is that it allows the graphics driver to optimize the code specifically for the device it's running on. The next step is setting the uniform parameters, and the speaker demonstrates how to do this by obtaining the location of the variable and then setting its value.
* [**00:45:00**](https://youtube.com/watch?v=je_PhVKJYng&t=2700) In this section, the speaker explains the process of setting up a WebGL program and modifying variables within it. They emphasize the importance of specifying which program they are using, as different shaders may be used for different objects in the scene. The speaker also introduces the naming convention for setting matrix and XYZ values, as well as passing arrays of values. They go on to discuss how the vertex attributes are passed to the vertex shader and the process of connecting and splitting vertex buffer objects in WebGL.
* [**00:50:00**](https://youtube.com/watch?v=je_PhVKJYng&t=3000) In this section, the speaker explains the process of rendering triangles in WebGL. They first retrieve the location of the position attribute and bind the position buffer to access the vertex buffer object in GPU memory. Then, they specify that the position attribute is a 3D vector of XYZ values. They also enable the attribute so that WebGL can use it. The speaker repeats the same process for the color attribute. After setting up the attributes, the speaker clears the screen, selects the program to be used for rendering, and finally initiates the draw call to begin rendering the triangles.
* [**00:55:00**](https://youtube.com/watch?v=je_PhVKJYng&t=3300) In this section, the instructor introduces the concept of vertex buffer objects and explains how to draw shapes using WebGL. He demonstrates a simple example of drawing a quad by drawing two triangles. The code for this example involves HTML initialization, shader code for the vertex and fragment shaders, and WebGL initialization and rendering. The instructor also explains how to set uniform variables and specify the usage of vertex buffer objects. Ultimately, this code will clear the canvas and draw the specified triangles.
* [**01:00:00**](https://youtube.com/watch?v=je_PhVKJYng&t=3600) In this section, the instructor discusses the html code necessary to create a canvas for WebGL rendering. He mentions that users can download the html page from the course website to see and experiment with the code. The instructor also provides a simplified view of the GPU pipeline for WebGL rendering, which includes components such as the API, vertex shader, primitive assembly, rasterizer, and fragment shader. He notes that there are more detailed representations of the pipeline available, but he prefers the simpler conceptual understanding. The section concludes with the instructor expressing his happiness that viewers have stayed till the end and mentioning that the next topic will be curves.

**Summary of Intro to Graphics 09 - Curves (Part 1)**

00:00:00 - 01:00:00

In this YouTube video titled "Intro to Graphics 09 - Curves (Part 1)", the lecturer introduces the concept of curves in computer graphics and discusses their representation and applications. They explain the importance of curves in representing thin and long objects as well as for animation and graph displays. The lecturer discusses various ways to represent curves, including implicit and parametric representations. They also explain the concept of re-parameterization and its effect on the curve. The video covers the representation of curves using polynomials and introduces the concept of bezier curves and their control points. The importance of control points in defining curves and their relationship to transformations is also discussed. The speaker mentions the Casteljau algorithm for finding points on bezier curves and the subdivision process for splitting curves. They also touch upon piecewise polynomial curves as an alternative for longer curves. Overall, the video provides an introduction to the topic of curves in computer graphics and lays the foundation for further discussions on more complex curves.

00:00:00 In this section, the lecturer introduces the topic of curves in computer graphics, discussing their importance and applications. Curves are used to represent thin and long objects like hair and cloth, as well as for animation and graph displays. The lecturer mentions that many 3D models are prepared using curves, even if the end result is not necessarily a curved model. The concepts of curves in 2D are explained, with the lecturer giving examples of linear functions to demonstrate the relationship between x and y coordinates. The simplicity and versatility of curves are emphasized, setting the foundation for further discussions on more complex curves.

00:05:00 In this section, the speaker discusses the concept of curves and their functional representation. They explain that every curve can be represented by a function where each x value corresponds to one and only one y value. However, if a curve is rotated, it no longer qualifies as a function since one x value can have multiple y values. In such cases, an implicit representation is used, which is a function of x and y that equals zero. The implicit representation allows for more complex curves like circles and can be expressed using vector notation for simplicity.

00:10:00 In this section, the speaker discusses the concept of curves and different ways to represent them in computer graphics. They mention that while implicit representations are useful for certain situations, they can be inconvenient when it comes to drawing curves. Instead, the speaker introduces the idea of a parametric representation, where a function can give the position on the curve based on a scalar parameter value. This representation is more efficient and widely used in computer graphics as it directly provides the solution without requiring the equation to be solved. The speaker also mentions that the choice of parameter values can vary and will affect the function, but the same curve shape can be achieved regardless.

00:15:00 In this section, the speaker discusses the concept of re-parameterization of curves and how it affects the function and speed at different points on the curve. They explain that re-parameterization changes how the T-value is interpreted along the curve, without altering its shape. They emphasize the importance of understanding the independent variable T and its role in the representation of curves. They then introduce a specific example of a line and its implicit and parametric representations, highlighting the difference in equations between the two representations.

00:20:00 In this section, the speaker discusses parametric representations and the benefits of using them. They use the example of a line that goes through two points, expressing it as a vector function. They explain that the parametric representation includes a point on the line and a direction, allowing for easy manipulation of the line by changing the parameter value. The speaker also shows an alternative polynomial representation of the equation. Overall, the parametric representation is seen as helpful and intuitive for understanding and working with lines.

00:25:00 In this section, the speaker discusses how lines can be represented as polynomials of degree one, which is a linear interpolation between two points. The weights of the points, p0 and p1, always add up to one. This understanding is crucial for grasping more complex curves that don't resemble line segments. The speaker then introduces second degree polynomials, which form the shape of a parabola. The polynomial form of a parabola is expressed as a t squared plus bt plus c, where a, b, and c are vectors. However, this representation may not be very intuitive, so alternative representations will be explored.

00:30:00 In this section, the speaker discusses the representation of curves and introduces a new way to write the equation of a quadratic curve. They explain that while it is useful to make predictions based on a parameter value, it is not intuitive or obvious how to draw the curve. To overcome this, they propose using tangents and points on the curve to create a more helpful representation. By finding the intersection of the tangents, they name these points P0, P1, and P2. Using these points, they derive the equation for a quadratic curve in the form P0 - 2P1 + P2. They also introduce a cleaner way of writing the equation using the form 1 - t^2 + 2(1 - t)tP1 + t^2P2, which allows for easy curve creation based on three points.

00:35:00 In this section, the speaker discusses the bezier form of a quadratic polynomial and its control points. They explain that bezier curves are essentially polynomial curves written in a special form where each control point appears only once. The speaker also mentions that bezier curves can be represented in higher degrees, such as cubic bezier curves with four control points. They note that while quadratic bezier curves are planar curves, cubic bezier curves can exist in three dimensions. This versatility in representation makes the cubic bezier curve formulation the most commonly used in computer graphics, especially for 3D applications.

00:40:00 In this section, the speaker explains that bezier curves can be written using polynomial functions, with cubic bezier curves being the most commonly used. Higher degree polynomials and bezier curves can also be used, but cubic curves are preferred for most purposes. However, bezier curves can only represent polynomial functions and cannot accurately represent shapes like circles. Bezier curves interpolate the first and last control points, but not the intermediate control points. They are contained within the convex hull of the control points, allowing for easy evaluation of the curve's closeness. Additionally, bezier curves are affine invariant, meaning that if a transformation is applied to the control points, the curve will transform accordingly.

00:45:00 In this section, the speaker discusses the importance of control points in defining curves and how applying transformations to control points does not change the shape of the curve. This property allows for easy transformation of curves without the need to generate a new curve. The speaker also introduces the concept of derivatives in calculus and explains how the derivative of a parametric curve gives the direction of the curve. They demonstrate this concept by taking the derivative of a cubic Bezier curve, resulting in a quadratic Bezier curve representing the directions along the original curve.

00:50:00 In this section, the speaker discusses the relationship between cubic Bezier curves and quadratic Bezier curves. They explain that by taking the integral of a curve, they can generate a curve, although not exactly due to a constant term. They also mention the derivative relationship between these curves, with the second derivative resulting in a line. The speaker credits Pierre Bézier, a French engineer at Renault, for developing these curves to represent car shapes. Additionally, they mention another French engineer, Paul de Casteljau, who had a similar idea around the same time. The speaker then introduces the consensus algorithm for drawing Bezier curves, explaining how to split line segments proportionally using a parameter "t" to create points that form the curve.

00:55:00 In this section, the speaker explains how to find a point on a curve using the Casteljau algorithm. They mention that the algorithm, although credited to Casteljau, is commonly used with Bezier curves. The speaker also discusses the subdivision process of splitting a cubic Bezier curve into two, by using control points. They explain that this splitting algorithm is used in many graphics algorithms for handling Bezier curves. The speaker then introduces piecewise polynomial curves as an alternative to using a large number of control points with Bezier curves. By using cubic polynomials and adding multiple pieces together, longer curves can be formed. Overall, the speaker emphasizes the importance of the Casteljau algorithm and the versatility of Bezier curves in various graphics applications.

01:00:00 - 01:05:00

In this video, the speaker explains the concept of curve continuity in graphics. They discuss different types of continuity, including C0, C1, C2, and G1, and how they are achieved by manipulating the control points and derivatives of the curves at the joining points. The speaker also highlights the importance of curvature continuity and its relationship with C2 continuity. They mention that in computer graphics, emphasis is primarily placed on C0, C1, and C2 continuity, as higher levels become difficult to determine visually. Finally, they introduce the concept of piecewise polynomial curves and how they allow for achieving different continuity levels.

01:00:00 In this section, the speaker discusses different types of curve continuity, starting with c0 continuity, where two curves are touching at the joining point. They explain that c0 continuity is achieved by using the same control point for both curves, making them essentially one curve. They then move on to c1 continuity, which focuses on the direction of the curve at the joining point. If the derivative of both curves is the same at that point, the curve has c1 continuity. The speaker mentions that they may not necessarily care about the magnitude of the derivative, as it depends on the parameterization of the curve. Next, they introduce g1 continuity, where the derivatives on either side of the joining point may be different in magnitude but have the same direction. They explain that g1 continuity can be converted to c1 continuity by reparameterizing the curve. The speaker also briefly discusses c2 continuity, where the change in derivative direction is continuous, and g2 continuity, which concerns the continuity of the second derivative direction. Finally, they mention the importance of curvature continuity, which is measured by the radius of a circle that fits the curve at the joining point, and how it relates to c2 continuity.

01:05:00 In this section, the speaker discusses the concept of continuity in curves and how it relates to graphics. They explain that while the bending of a curve can be seen visually, beyond a certain level of continuity, such as C3 or G3, it becomes difficult to determine visually. Therefore, in computer graphics, they primarily focus on C0, C1, and C2 continuity. The speaker then introduces the idea of piecewise polynomial curves, where each piece of the curve is defined by a set of control points. They explain that this allows them to avoid using high-degree polynomials and achieve different levels of continuity. The speaker concludes by mentioning that in the next lecture, they will explore different curve formulations to create piecewise continuous curves with desired properties

**Curve 2**

00:00:00 - 00:55:00

In this section of the video, the lecturer discusses different aspects of curves, including parametric representation, Bezier curves, B-spline curves, rational curves, NURBS surfaces, and interpolating curves. They explain the importance of properly defining control points and weights to achieve the desired curve shape. The lecturer also introduces different parameterization methods and discusses the use of interpolation functions and blending functions to create smooth curves. They conclude by mentioning that this lecture covers the most important curve formulations in computer graphics and invite the viewer to continue learning about surfaces in the next lecture.

00:00:00 In this section, the lecturer continues the discussion on curves, specifically focusing on parametric representation versus implicit representation. They review the previous lecture and discuss how function representation is not helpful for curves. Instead, they explain how parametric representation is more useful and give an example of a curve represented by a parameter value. The lecturer then introduces Bezier curves and explains how they can be quadratic or cubic depending on the number of control points. They discuss the flexibility of cubic Bezier curves in shaping 3D curves. The lecturer also mentions the derivatives of Bezier curves, showing how the derivatives form quadratic and linear Bezier curves.

00:05:00 In this section, the speaker talks about how curves can be constructed using piecewise Bezier curves, where each small piece is defined by a polynomial curve. However, this method requires defining a lot of control points. The speaker then discusses different formulations for generating curves and how the shape of the curve can vary depending on the control points and formulation used. They also explain the concept of reparameterization, where different polynomials can produce the same curve shape.

00:10:00 In this section, the video discusses how changing the speed at which the parameter moves along the curve does not alter the shape of the curve. The video explains that instead of changing the degree of the curve, it is generally more common to shift the curve's starting and ending points or manipulate the parameter values to achieve different representations of the same curve. The concept of parameterization is introduced, where the parameter values are determined to correspond to specific points on the curve. The video emphasizes that the choice of parameter values can affect the appearance of the curve, but ultimately, it is up to the user's discretion. The discussion then transitions to a general form of the Bezier curves, which are defined by control points and coefficients.

00:15:00 In this section, the concept of Bezier formulation is explained, which is essentially a weighted average of control points to produce a curve. The weights depend on a parameter value, and changing these weights can result in different curve formulations. The important requirement is that the weights must sum up to one, known as the partition of unity. If the weights do not add up to one, the resulting curve can behave unexpectedly and lose affine invariance. Therefore, it is crucial to ensure that the weights are properly defined to maintain the desired properties of the curve. Additionally, it is noted that having weights that do not sum up to one introduces a hidden control point at the origin.

00:20:00 In this section, the speaker explains the importance of having weights that add up to one when formulating a curve as a weighted sum of control points. They introduce the concept of B-spline curves, which are defined by different orders of polynomials and can provide higher levels of continuity. The speaker shows an example of a piecewise cubic curve, explaining that each piece is defined by four control points but may not necessarily interpolate them. However, they demonstrate how a B-spline curve can be forced to interpolate the first and last control points by repeating them multiple times depending on the order of the curve. The speaker emphasizes that B-spline curves are widely used in computer graphics.

00:25:00 In this section, the speaker discusses how to come up with different curve formulations by specifying a way to generate weights that add up to one. They explain that cubic spline curves can be converted to cubic Bezier curves, although the control points used will be different. The importance of partition of unity is emphasized, with the suggestion of normalizing weights to create rational curves. Rational curves are commonly used in computer graphics, with the ability to define different weight parameters for each control point. The speaker shows an example of a rational quadratic Bezier curve with additional weights and suggests that adjusting one weight can result in different curve shapes.

00:30:00 In this section, the speaker explains how different weight values affect the shape of curves. A weight value greater than one will produce hyperbolas, while a value smaller than one will result in conics like ellipses. The speaker also mentions that by setting the weight value to zero, the curve becomes a linear interpolation between the control points. Negative weight values cause the curve to bend in the opposite direction. The concept of rational Bézier curves is introduced, where an additional weight parameter is added to each control point, allowing for the generation of exact ellipses and circles. The speaker then explains that the same concept can be applied to B-splines, resulting in non-uniform rational B-splines (NURBS), commonly used in computer graphics to model curves and surfaces.

00:35:00 In this section, the speaker discusses the use of curves for generating surfaces, specifically NURBS surfaces. While NURBS surfaces are not as popular today as they once were, they are still used for various purposes. One drawback of NURBS surfaces is that they do not interpolate the control points, which means that the surface does not directly follow the control polygon. However, despite this drawback, NURBS surfaces have other favorable properties and can be used to create highly detailed models. The speaker also mentions that NURBS curves are often considered "approximating curves" because they do not precisely go through all the control points and may only interpolate the first and last control points. Nonetheless, if precise control point interpolation is required, other methods may need to be explored.

00:40:00 In this section, the speaker discusses interpolating curves, which are curves that pass through all of their control points. A popular example of an interpolating curve is the Catmull-Rom curve, which interpolates the two middle control points but not the first and last control points. The speaker explains that although the first and last control points are not interpolated, they can be replicated and used in the curve calculation, ensuring that the curve still passes through those points. The Catmull-Rom curve can be converted into a Bezier curve, but the control points will look slightly different. The speaker also introduces the Varian-Baron-Goldman formulation of the Catmull-Rom curve, which involves interpolating neighboring control points to generate the curve.

00:45:00 In this section, the speaker discusses different parameterizations for copper curves. They explain that uniform parametrization is where the control points are spaced uniformly in parameter space. However, this method does not produce good curves. Another option is chordal parametrization, where the parameter values are spaced proportional to the distance between the control points. This also has its drawbacks. The speaker then introduces centripetal parameterization, where the parameter values are proportional to the square root of the distance between control points. This parameterization method will be further explored in the upcoming discussion.

00:50:00 In this section, the instructor discusses the distance between control points and its significance in curve parameterization options. They explain that while it may not be obvious initially, through polynomial formulation, it becomes clear that the centripetal parameterization option is likely the best one. This is because it minimizes self-intersections and keeps the curve close to the control polygon, providing better control over the curve. The instructor also mentions that most curve formulations in computer graphics are polynomials or rational polynomials, but introduces a different curve formulation called the class of Ct interpolating curves, which offers C2 continuity and has nice properties. These curves are formed by an interpolation function and a blending function, with the interpolation function defining a curve that passes through three consecutive control points.

00:55:00 In this section, the speaker discusses the process of reparameterizing curves, as well as the use of interpolation functions and blending functions to create smooth curves. They explain that the particular blending function they use, which involves cosine squared and sine squared, allows for C2 continuity in the resulting curve. The speaker also mentions different examples of interpolation functions, including quadratic Bezier curves and elliptical formulations. They provide further details and recommend watching a recorded talk on their YouTube channel for a more in-depth understanding of these curves. The section concludes with an example of a 3D curve formed using these interpolation functions. The speaker states that while this lecture does not cover all curve formulations in computer graphics, it includes the most important ones, including the latest one introduced. They invite the viewer to join the next lecture on surfaces.