FLAME UNIVERSITY

EVERLASTING learning

FUNDAMENTALS OF COMPUTER GRAPHICS (CSIT304)

BEYOND BASIC 3D GRAPHICS

CHIRANJOY CHATTOPADHYAY

Associate Professor,

FLAME School of Computation and Data Science

INTRODUCTION

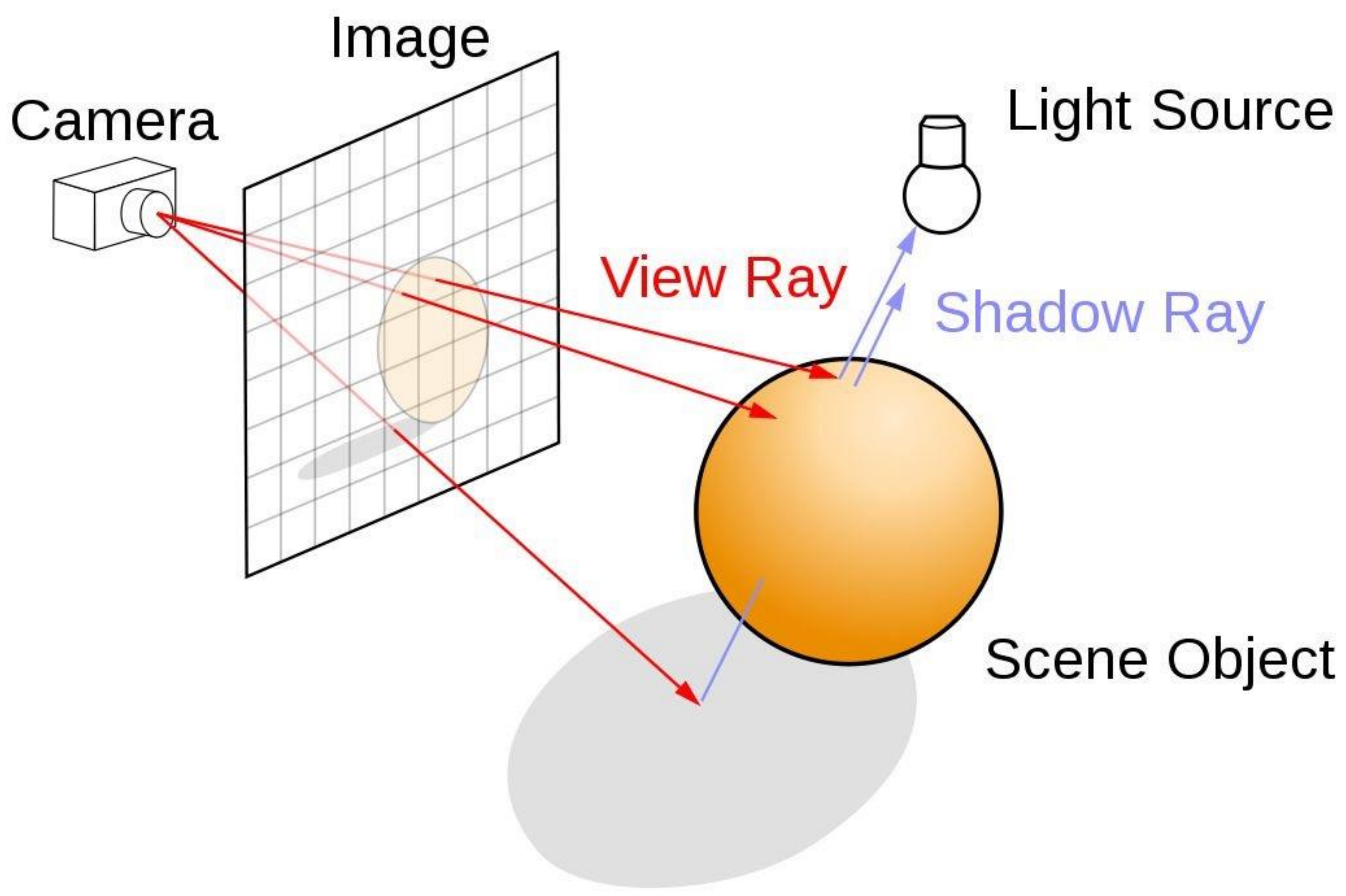
- Graphics systems in which an image can be generated in a fraction of a second
- Real-time graphics still can't match the realism of the very highest quality computer graphics, such as what can be found in movies
- Brief look at some techniques that can be used for very high quality graphics.
- Technique for higher quality graphics

RAY TRACING

OVERVIEW

- A technique to generate realistic images by simulating the behavior of light in a 3D scene.
- Rays of light are traced from the viewer's eye through each pixel of an image
- Then traced back into the scene to determine the color and intensity of the light for that pixel.
- This process of tracing rays of light can be computationally expensive
    - Simulating the interaction of light with objects in the scene, reflection, refraction, and shadowing.
- Highly valued for creating highly realistic images with accurate lighting and shadows
- A popular technique in fields such as film, video game design, and architecture visualization.

ILLUSTRATION



RAY TRACING FUNDAMENTALS

- Ray casting
- Path tracing
- Bounding volume hierarchy
- Denoising filtering

RAY CASTING

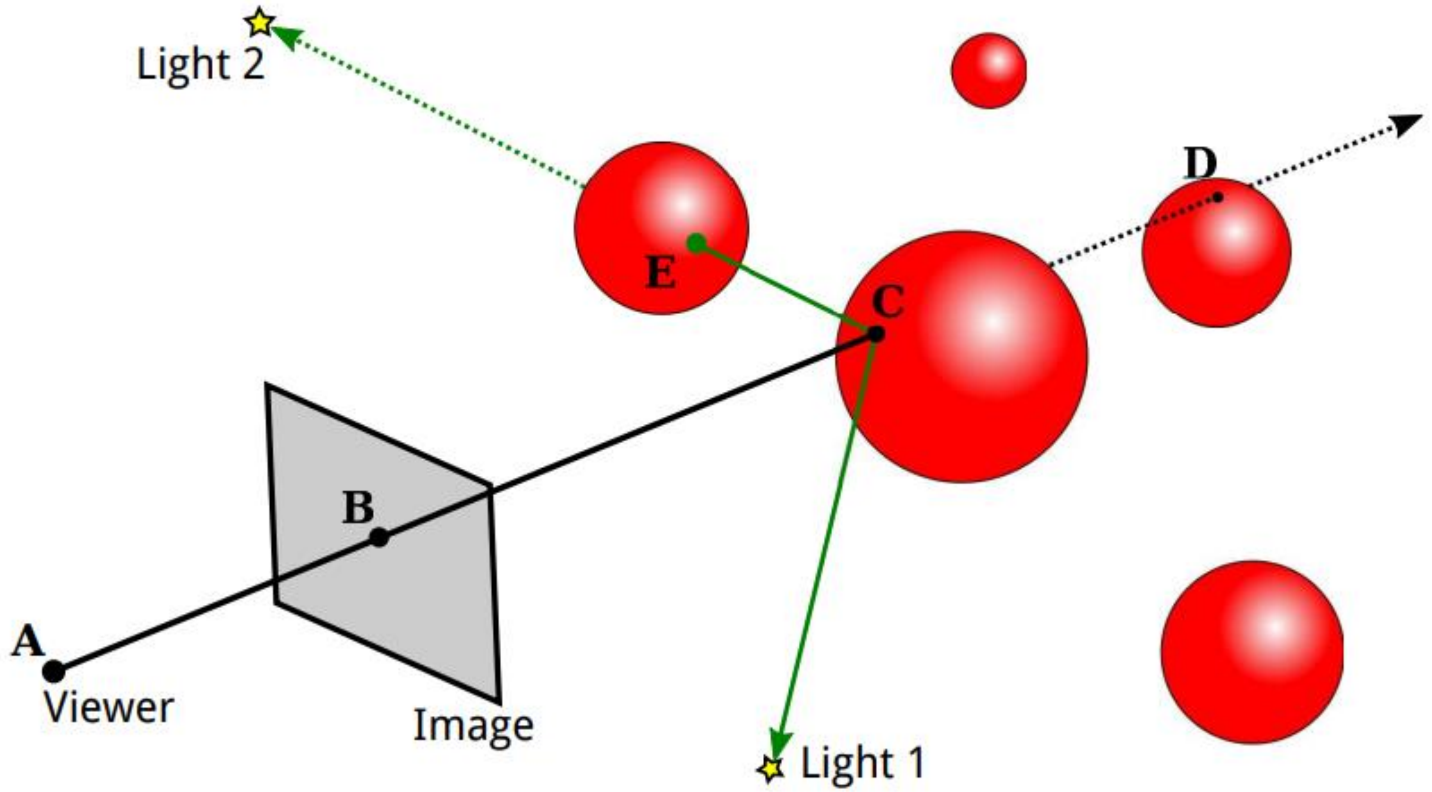
RAY CASTING

- The process in a ray tracing algorithm that shoots one or more rays from the camera (eye position) through each pixel in an image plane
    - Then tests to see if the rays intersect any primitives (triangles) in the scene.
- If a ray passing through a pixel and out into the 3D scene hits a primitive
    - Then the distance along the ray from the origin (camera or eye point) to the primitive is determined,
    - The color data from the primitive contributes to the final color of the pixel.
- The ray may also bounce and hit other objects and pick up color and lighting information from them.

RAY CASTER

- A Ray caster takes an initial point and a direction, given as a vector.
- The point and vector determine a ray, that is, a half-infinite line that extends from a starting point, in some direction, to infinity.
- The Ray caster can find all the intersections of the ray with a given set of objects in a scene, sorted by order of distance from the rays's starting point.
- We are interested in the first intersection, the one that is closest to the starting point.

ILLUSTRATION



IMPLEMENTING RAY CASTING

Computational complexity

- Can be computationally expensive
- Scene contains a large number of objects or complex geometry.
- Can result in slow rendering times
- A significant drawback in real-time applications such as video games.

Limited accuracy

- If the number of rays cast is insufficient
- If the scene contains objects that are too small
- Too close together to be accurately represented by the available grid resolution.
- Lead to artifacts such as aliasing, banding, or inaccurate shadows.
- Interacting with triangles

Occlusion

- Objects in the scene can block the path of rays
- Leading to incomplete or incorrect rendering.
- Can be addressed using techniques such as shadow mapping or ray tracing
- These methods can add additional computational overhead.

Memory requirements

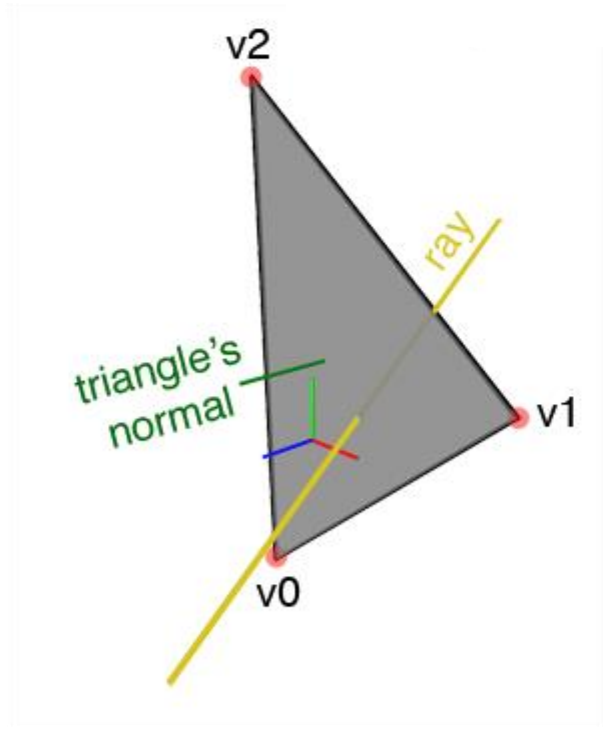
- Can require significant amounts of memory to store the scene geometry and intermediate results.
- This can be a limitation in applications with limited memory
- E.g., mobile devices or embedded systems.

INTERACTION WITH TRIANGLES

- It may seem straightforward to solve ray-triangle intersections in theory
- The complexity arises from the various cases that need to be considered.
- The challenge lies in writing a robust routine that can efficiently handle all these cases.
- This can be a difficult task that requires careful attention to detail and optimization.

GEOMETRIC PRIMITIVES

- A geometric primitive represents a 3D object in a rendering program
- A triangle is not a geometry type of its own.
- It is rather a subset of the polygon primitive type.



BARYCENTRIC COORDINATES

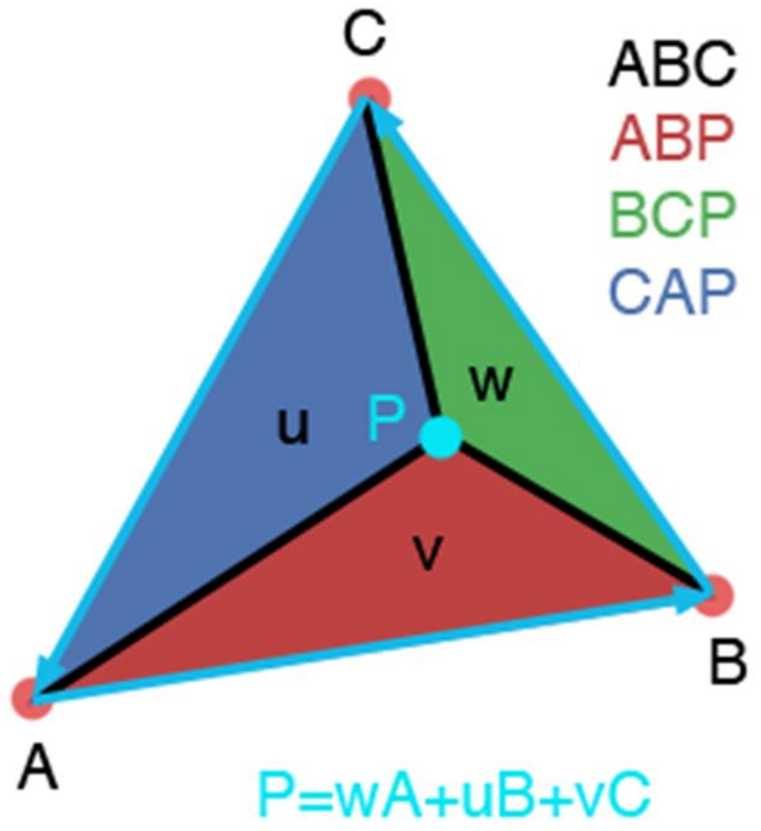
- Barycentric coordinates are particularly important in CG.
- Barycentric coordinates can be used to express the position of any point located on the triangle with three scalars.
- The location of this point includes
    - any position inside the triangle,
    - any position on any of the three edges of the triangles, or
    - any one of the three triangle's vertices themselves.

CALCULATION

$$P = uA + vB + wC.$$

- $A, B,$ and C are the vertices of a triangle
- $u, v,$ and w (the barycentric coordinates),
- Three real numbers (scalars) such that $u + v + w = 1$ (barycentric coordinates are normalized).
- From two of the coordinates we can find the third one: $w = 1 - u - v$
- We can establish that $u + v \leq 1$

ILLUSTRATION

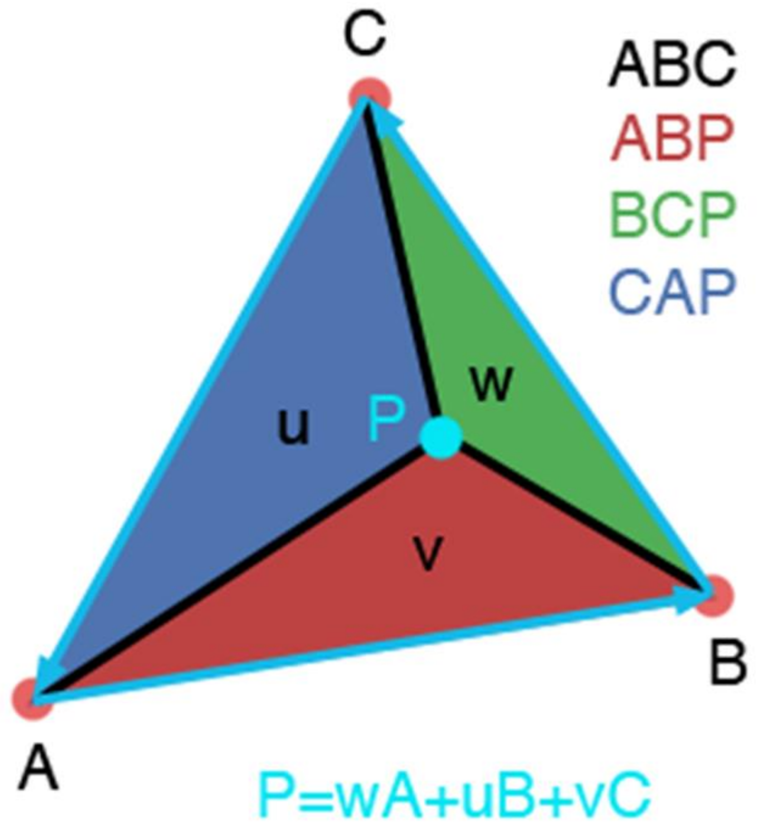


ALTERNATE NOTATION

- $P = A + u * AB + v * AC$
- $u \geq 0, v \geq 0, u + v \leq 1$
- Interpretation: *starts from A, move a little bit in the direction of AB, then a little bit in the direction of AC and you will find P*

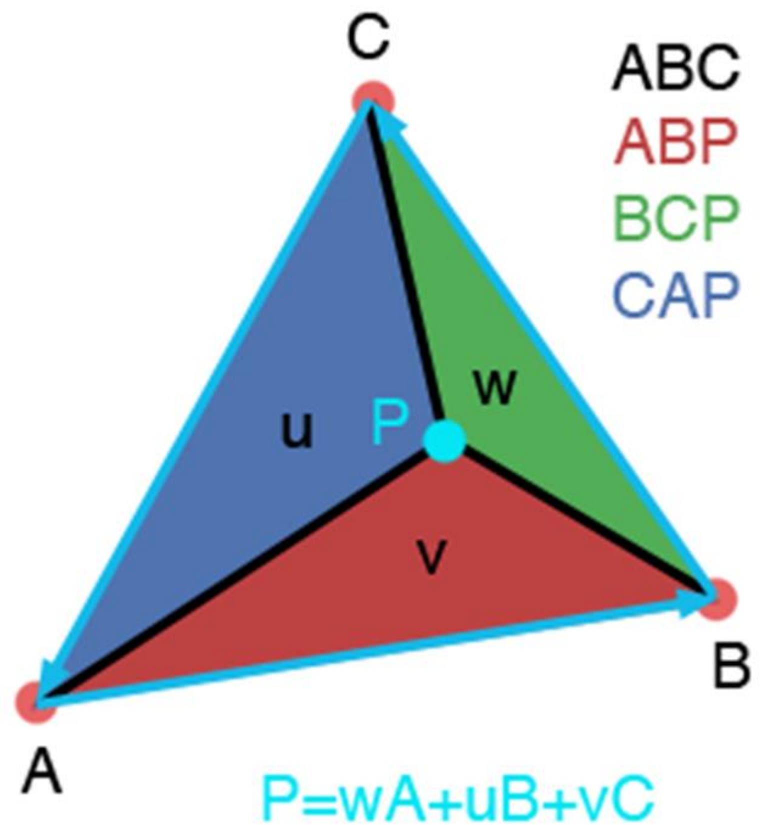
BARYCENTRIC COORDINATES

- Barycentric coordinates are also known as areal coordinates.
- Indicates that the coordinates u, v, and w are proportional to
  - the area of the three sub-triangles defined by P,
  - the point located on the triangle, and
  - the triangle's vertices (A, B, C).
- These three sub-triangles are denoted ABP, BCP, and CAP



FORMULA

- Formulas used to compute the barycentric coordinates:



$$u = \frac{TriangleCAP_{Area}}{TriangleABC_{Area}}$$

$$v = \frac{TriangleABP_{Area}}{TriangleABC_{Area}}$$

$$w = \frac{TriangleBCP_{Area}}{TriangleABC_{Area}}$$

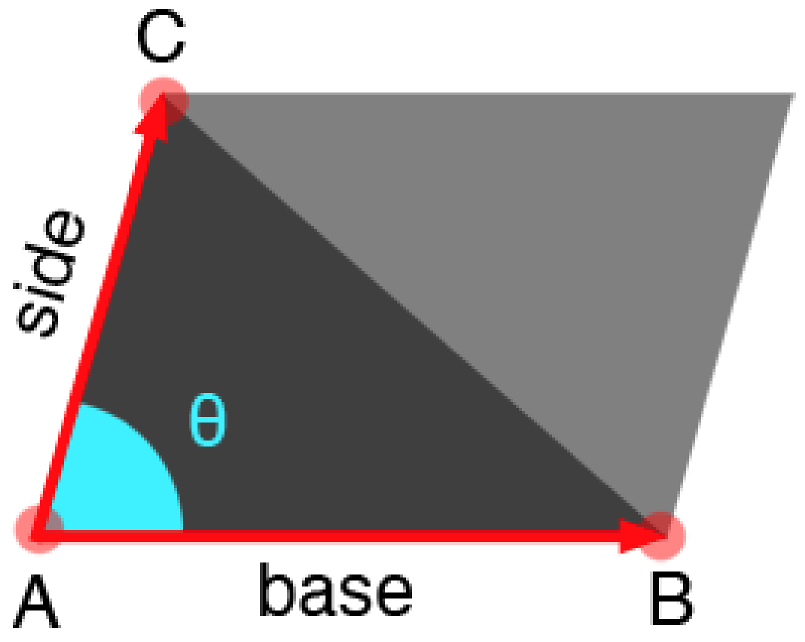
AREA OF A TRIANGLE CALCULATION GIVEN THE VERTICES

- To calculate the area of a triangle given its three vertices (x1,y1), (x2,y2), and (x3,y3), following is the formula for the area of a triangle formed by three points in the Cartesian coordinate system.
- Here's the formula:

$$\text{Area} = \frac{1}{2}\left|x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)\right|$$

COMPUTING THE BARYCENTRIC COORDINATES

- Computing the area of the triangle
- Duplicate the triangle and mirror it along its longest edge, you get a parallelogram.
- To compute the area of a parallelogram
  - Compute its base, and its side
  - Multiply these two numbers together scaled by $\sin(\theta)$
  - θ is the angle subtended by the vectors AB and AC
- The area of one triangle is half the area of the parallelogram.

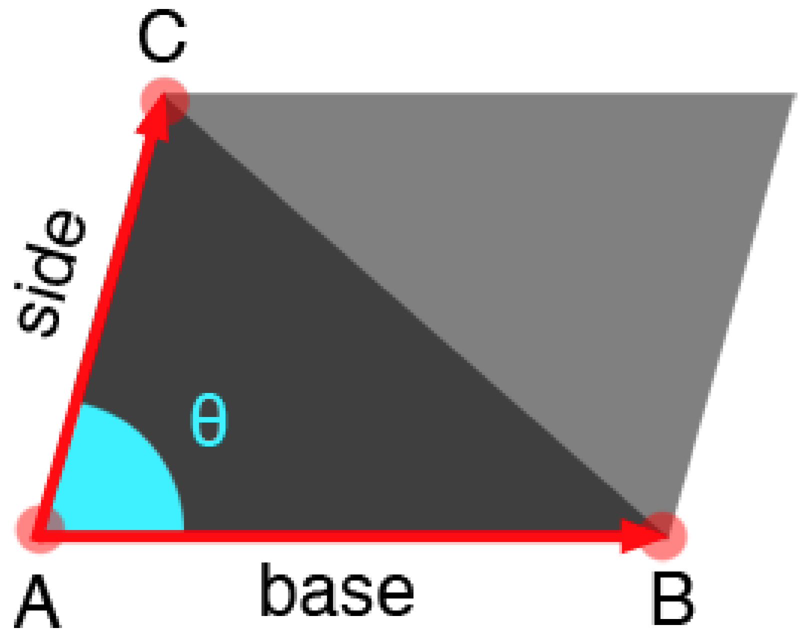


COMPUTING THE BARYCENTRIC COORDINATES

$$Triangle_{area} = \frac{||(B-A)|| * ||(C-A)|| sin(\theta)}{2}$$

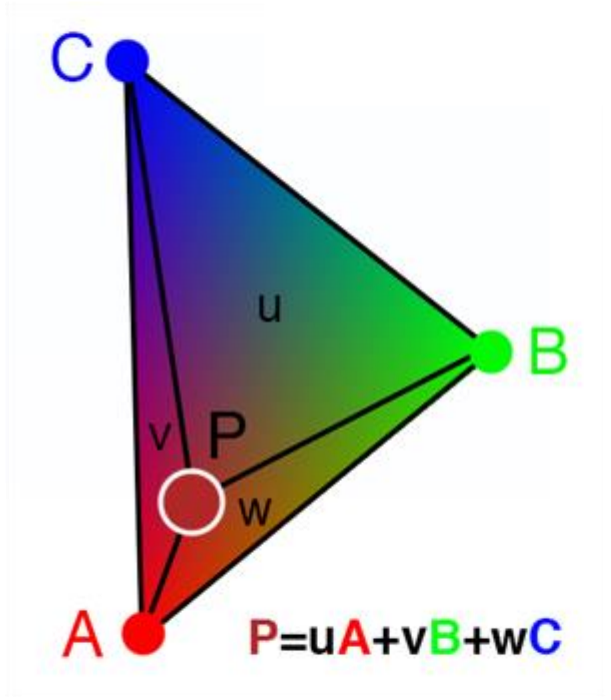
$$Parallelogram_{area} = ||(B-A) \times (C-A)||$$

$$Triangle_{area} = \frac{Parallelogram_{area}}{2}$$



USING BARYCENTRIC COORDINATES

- Barycentric coordinates are most useful in shading.
- A triangle is a flat surface and we can associate any additional information or data (points, color, vectors, etc.) to each one of its vertices.
- This information is usually called **vertex data.**



NUMERICAL EXAMPLE

- Consider a triangle with vertices A(1,1), *B*(3,4), and *C*(5,2). Let's say we have a point *P* inside this triangle, and we want to express its coordinates in terms of barycentric coordinates.

NUMERICAL EXAMPLE

- Consider a triangle with vertices A(1,1), *B*(3,4), and *C*(5,2). Let's say we have a point *P* inside this triangle, and we want to express its coordinates in terms of barycentric coordinates.

Let *P* have barycentric coordinates (*u*,*v*,*w*), where *u*, *v*, and *w* are the weights assigned to the vertices *A*, *B*, and *C* respectively.

$$P=uA+vB+wC$$

NUMERICAL EXAMPLE

- Consider a triangle with vertices A(1,1), *B*(3,4), and *C*(5,2). Let's say we have a point *P* inside this triangle, and we want to express its coordinates in terms of barycentric coordinates.

$$u+v+w=1$$

Now, let's say *P* has Cartesian coordinates (2,2). We want to find the barycentric coordinates (*u*,*v*,*w*) for this point.

$$\begin{aligned} 2 &= u \cdot 1 + v \cdot 3 + w \cdot 5 \\ 2 &= u \cdot 1 + v \cdot 4 + w \cdot 2 \\ u + v + w &= 1 \end{aligned}$$

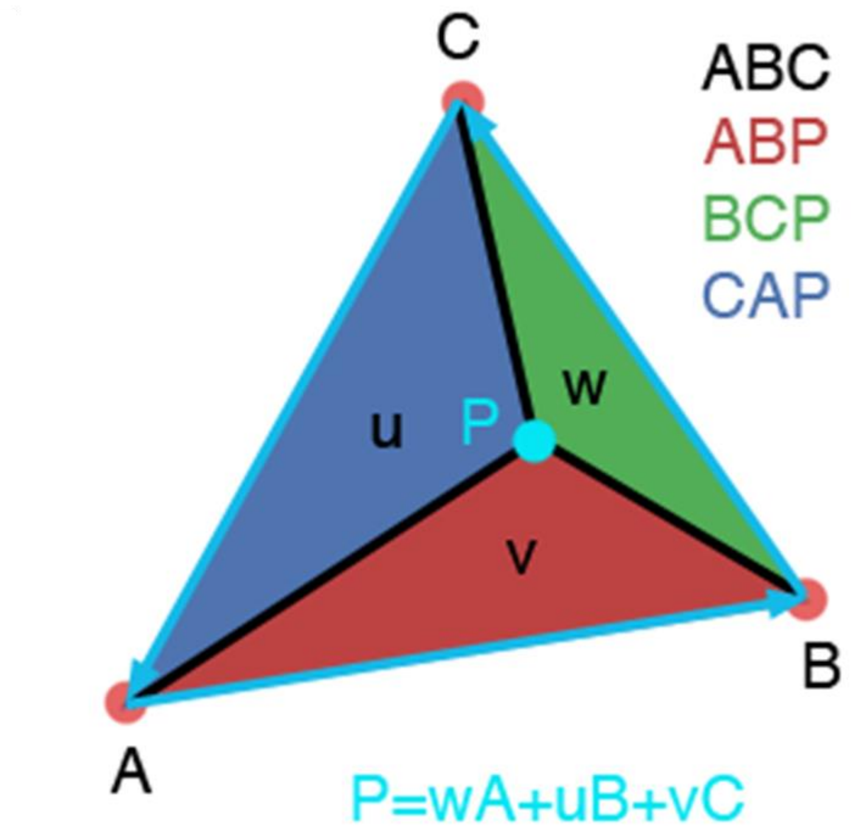
INSIDE OUTSIDE TEST USING BARYCENTRIC COORDINATES

- To determine whether a point is inside a triangle or not using Barycentric coordinates, you can follow these steps:

1. **Calculate Barycentric Coordinates**: Compute the Barycentric coordinates (u,v,w) of the point with respect to the vertices of the triangle.
2. **Check Barycentric Coordinate Range**: Ensure that all Barycentric coordinates are within the range [0,1][0,1].
3. **Inside the Triangle**: If all Barycentric coordinates are within the range [0,1] and their sum is equal to 1, then the point lies inside the triangle.

INSIDE OUTSIDE TEST USING BARYCENTRIC COORDINATES

- Formulas used to compute the barycentric coordinates:



$$u = \frac{TriangleCAP_{Area}}{TriangleABC_{Area}}$$

$$v = \frac{TriangleABP_{Area}}{TriangleABC_{Area}}$$

$$w = \frac{TriangleBCP_{Area}}{TriangleABC_{Area}}$$

ALGORITHM

```
// vertex position

Vec3f triVertex[3] = {{-3,-3,5}, {0,3,5}, {3,-3,5}};

// vertex data

Vec3f triColor[3] = {{1,0,0}, {0,1,0}, {0,0,1}};

if (rayTriangleIntersect(...)) {

    // compute pixel color

    // col = w*col0 + u*col1 + v*col2 where w = 1-u-v

    Vec3f PhitColor = u * triColor[0] + v * triColor[1] + (1 - u
- v) * triColor[2];

}
```

MÖLLER-TRUMBORE ALGORITHM

- It is a fast ray-triangle intersection algorithm
- Takes advantage of the parameterization of P
- The intersection point in barycentric coordinates

$$P = wA + uB + vC$$

- $w = 1 - u - v$
- $P = O + tD$
- Represent the above equation using matrix notation

CRAMER'S RULE

$$\begin{aligned} 2x + 1y + 1z &= 3 \\ 1x - 1y - 1z &= 0 \\ 1x + 2y + 1z &= 0 \end{aligned}$$

APPLICATION OF CRAMER'S RULE TO FIND UNKNOWNS

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{[|-D \quad E_1 \quad E_2|]} \begin{bmatrix} |T \quad E_1 \quad E_2| \\ |-D \quad T \quad E_2| \\ |-D \quad E_1 \quad T| \end{bmatrix}$$

$$T = O - A$$

$$E_1 = B - A$$

$$E_2 = C - A$$

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (T \times E_1) \cdot E_2 \\ (D \times E_2) \cdot T \\ (T \times E_1) \cdot D \end{bmatrix} == \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix}$$

$$P = (D \times E_2)$$

$$P = (T \times E_1)$$

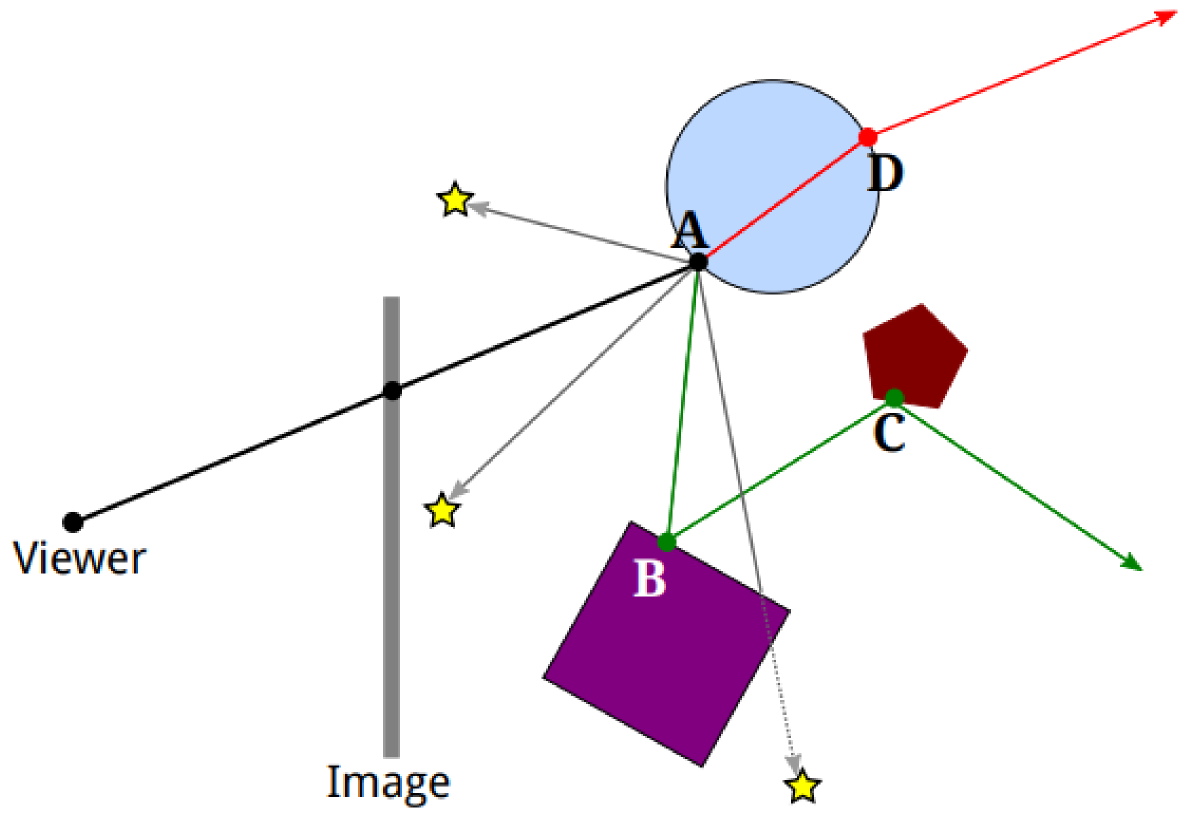
RECURSIVE RAY TRACING

- Basic ray casting can be used to compute rendering with the addition of shadow rays, to implement shadows as well.

- More features can be implemented by casting a few more rays.

- The improved algorithm is called **ray tracing**.

ILLUSTRATION



COMPUTATION

- The goal is to compute a color for a point on an image.
- We cast a ray from the viewpoint through the image and into the scene,
- Determine the first intersection point of the ray with an object.
- The color of that point is computed by adding up the contributions from different sources.

EFFECT OF REFLECTION

- There is diffuse, specular, and possibly ambient reflection of light from various light sources.
- These contributions are based on the diffuse, specular, and ambient colors of the object, on the normal vector to the object, and on the properties of the light sources.
- Some color properties of the object, usually the ambient and diffuse colors, might come from a texture.
- The specular contribution can produce specular highlights, which are essentially the reflections of light sources.
- Shadow rays are used to determine which directional and point lights illuminate the object

EFFECT OF SURFACE TYPE

- If the surface has mirror-like reflection, then a reflected ray is cast and ray tracing is applied recursively to find a color for that ray.
- The contribution from that ray is combined with other contributions to the color of the surface, depending on the strength of the mirror reflection.
- For a perfect mirror, the reflected ray contributes 100% of the color, but in general the contribution is less.
- Mirror reflectivity is a new material property.
- It is responsible for reflections of one object on the surface of another, while specular color is responsible for specular highlights.

EFFECT OF TRANSPARENCY

- If the object is translucent, then a refracted ray is cast, and ray tracing is used to find its color.
- The contribution of that ray to the color of the object depends on the degree of transparency of the object, since some of the light can be absorbed rather than transmitted.
- The degree of transparency can depend on the wavelength of the light—as it does, for example, in colored glass

RECURSIVE ALGORITHM

- The ray tracing algorithm is recursive
- Recursion needs a base case, to come a time when, instead of calling itself, the algorithm simply returns a value.
- A base case occurs whenever a casted ray does not intersect any objects.
- Another kind of base case can occur when it is determined that casting more rays cannot contribute any significant amount to the color of the image.
- A ray tracing algorithm should always be run with a maximum recursion depth, to put an absolute limit on the number of times the algorithm will call itself.

LIMITATIONS OF RAY TRACING

Computational complexity	• Computationally intensive process, requiring significant processing power and time to produce high-quality images. • Real-time ray tracing is still challenging to achieve, although advances in hardware and software are reducing this limitation.
Limited geometry complexity	• Works best with simple geometric objects such as spheres, planes, and cylinders. • Complex objects with many faces, curves, or detailed textures can be challenging to render accurately with ray tracing.
Limited dynamic scene complexity	• Struggles with dynamic scenes, especially those with many moving objects, as the calculations must be performed for each frame. • Typically not suitable for real-time applications such as video games.
Limited lighting complexity	• Does not handle complex lighting scenarios such as caustics, volumetric lighting, or subsurface scattering as efficiently as other techniques such as **radiosity**.
Limited scalability	• Can become prohibitively expensive for large scenes or when rendering at high resolutions. • Hybrid approaches that combine ray tracing with other rendering techniques to balance performance and quality.

PATH TRACING

INTRODUCTION

- Path tracing simulates how light behaves in a scene to create realistic images.
- Path tracing uses random sampling to generate an approximation of the final image.
- Path tracing is computationally expensive and uses techniques such as importance sampling and Russian roulette to reduce the number of samples needed.
- Path tracing can handle complex lighting scenarios and render physically accurate materials.
- Path tracing is widely used in film, video games, and scientific visualization, but is often used in combination with other techniques for real-time performance.

DIFFERENCE WITH RAY TRACING

Feature	Path Tracing	Ray Tracing
Algorithm	Monte Carlo method that simulates the path of light rays	Casts rays from the camera to determine color
Light Transport	Accounts for indirect lighting as rays bounce around the scene	Only considers direct lighting visible to a surface
Noise	More noisy due to Monte Carlo nature, requires more samples per pixel	Produces less noise, but may lack subtle lighting effects

BIDIRECTIONAL SCATTERING DISTRIBUTION FUNCTION

- The Bidirectional Scattering Distribution Function (BSDF) is a function that describes
    - how light is scattered when it interacts with a surface, taking into account the incoming and outgoing directions of the light.
- In path tracing, the BSDF is used to determine
    - the probability of light scattering in different directions when it interacts with a surface.
    - This information is then used to simulate the behavior of light in a 3D scene, allowing realistic images to be generated.
- The BSDF is typically represented as a mathematical function that
    - Takes as input the incoming and outgoing directions of the light,
    - As well as other surface properties such as its color and roughness.

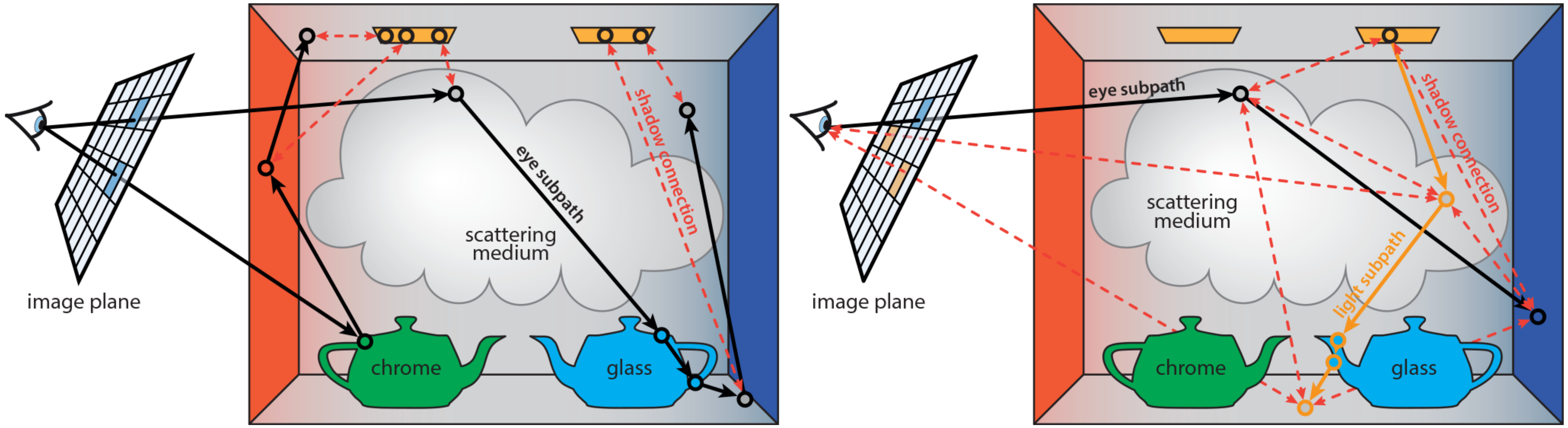
BIDIRECTIONAL SCATTERING DISTRIBUTION FUNCTION

- The BSDF is an important component in many rendering algorithms
    - Including path tracing,
    - As it allows for accurate simulation of light transport in a scene.
- There are many different types of BSDFs,
    - Each suited to different types of surfaces and materials.
    - For example, a glossy BSDF would be used for simulating reflections on a smooth, shiny surface, while a diffuse BSDF would be used for simulating light scattering on a rough, matte surface.

THE PATH TRACING ALGORITHM

1. **Camera Setup**
   - Determine the position, orientation, and parameters of the virtual camera that will capture the scene. This includes the resolution of the image and the field of view.
2. **Ray Generation**
   - For each pixel in the image, generate a primary ray that originates from the camera position and passes through the center of the pixel.
3. **Intersection Testing**
   - Trace the primary ray through the scene, testing for intersections with objects in the scene. This can be done using a bounding volume hierarchy or other acceleration structures.
4. **Surface Interaction**
   - If the primary ray intersects an object in the scene, compute the surface properties of the object at the intersection point. This includes the surface normal, texture color, and other material properties such as reflectivity or transparency.
5. **Random Walk**
   - Generate a new ray from the intersection point in a random direction based on the surface properties of the object. This is known as a random walk or path.
6. **Recursive Path Tracing**
   - Continue tracing the ray through the scene, performing intersection tests and surface interactions until the ray either hits a light source or reaches the maximum number of bounces allowed.
7. **Radiance Computation**
   - Compute the radiance or color of the pixel by accumulating the radiance along the path traced by the ray. This is done using the Monte Carlo integration method.
8. **Repeat:**
   - Repeat steps 2 through 7 for all pixels in the image until the desired level of convergence is reached.

ILLUSTRATION



Per H. Christensen, Wojciech Jarosz. The path to path-traced movies. Foundations and Trends in Computer Graphics and Vision, 10(2):103–175, October 2016.

BOUNDING VOLUME HIERARCHY (BVH)

- BVH is a data structure used in computer graphics for rendering and collision detection.
- BVH is a tree-like structure composed of bounding volumes that encapsulate objects or groups of objects in a scene.
- BVH reduces the number of geometric primitives that need to be processed by grouping them into larger volumes.
- To create a BVH, the scene is divided into bounding volumes using a recursive binary splitting approach.
- The BVH tree is constructed hierarchically, with each node representing a bounding volume enclosing the primitives in its child nodes.
- During rendering or collision detection, the BVH tree is traversed recursively to determine intersections with the view or collision volume.
- BVHs are widely used in video games, animation, and virtual reality to improve performance.

BVH VS OCTREE

Property	BVH	Octree
Type of data structure	Tree-like structure	Tree-like structure
Type of volumes used	Bounding volumes that can have arbitrary shapes	Axis-aligned boxes
Number of children per node	Two	Eight
How primitives are stored	Primitives are stored in leaf nodes	Primitives are stored in the smallest leaf nodes and their ancestors
Storage requirements	Lower storage requirements as bounding volumes can have arbitrary shapes	Higher storage requirements as axis-aligned boxes need to be stored for each node
Advantages	Can handle arbitrary shapes for bounding volumes, lower storage requirements, better performance for highly non-uniform distributions of primitives	More memory-efficient, can be more suitable for uniformly distributed primitives
Disadvantages	Higher construction time complexity, can be less memory-efficient for highly uniform distributions of primitives	Cannot handle arbitrary shapes for bounding volumes, more limited in the types of scenes it can efficiently handle

DENOISING FILTERING IN COMPUTER GRAPHICS

- Used in computer graphics to remove unwanted noise from images.
- Noise can occur due to various factors such as low light conditions, camera sensor limitations, and compression artifacts.
- Denoising filters analyze the image data and identify areas likely to contain noise.
- **Smoothing filters :** preserves the underlying structure of the image.
- **Bilateral filter:** takes into account both spatial and color information to preserve edges and details while removing noise.
- **Median filter:** Replaces each pixel with the median value of its neighboring pixels.
- Important for improving the visual quality of images in computer games, virtual reality, and film production.

FLAME UNIVERSITY

EVERLASTING learning

THANK YOU