Adicionei o Resumo do Nicolas do semestre passado no final do documento, pra caso alguém queira olhar o dele também. * Thiago

NÃO É PRA CAIR RASTERIZAÇÃO, MAS COMO TINHA UMA QUESTÃO SOBRE ISSO NA PROVA MODELO, VOU COLOCAR AQUI COISAS QUE EU ACHO QUE PODEM SER IMPORTANTES:

Rasterização:

Pontos Fracos:

- Todos os vértices são iluminados, mesmo que não sejam visíveis na imagem
- A iluminação de cada vértice é independente de todos os outros objetos do mundo: **Iluminação local**

Rasterização versus Ray Tracing

Rasterização	Ray Tracing

• Para cada **triângulo:**
  ○ Encontra os pixels que cobre
  ○ Para cada pixel: compara com o triângulo mais próximo até então
 	
Linear
	• Para cada **pixel:**
	  ○ Encontra o triângulo que pode ser o mais próximo
	  ○ Para cada triângulo: calcula a distância até o pixel
	
	Exponencial

http://slideplayer.com.br/slide/395462/

Slide 8 - iluminação

A luz é uma faixa de radiação eletromagnética visível ao olho humano e responsável pelo senso de visão.

Um **modelo de iluminação** descreve as entradas, as pressuposições, e as saídas usadas para calcular a iluminação da superfície dos elementos (cor e brilho). Esses modelos normalmente incluem **atributos da luz** (intensidade, cor, posição, direção e formato), **propriedades das superfície dos objetos** (cor, refletividade, transparência) e as interações entre os objetos e as luzes.

Os modelos podem ser divididos em categorias como:

- **Baseados em física:** modelos baseados em física real, fazem poucas pressuposições e necessitam de uma entrada de dados muito precisa, são usados em simulações e **levam um longo tempo para serem calculados.**
- **Não baseados em física:** são modelos que **enfatizam a velocidade e eficiência** no uso dos recursos computacionais, apenas necessitam que o resultado seja bom o suficiente para a aplicação alvo (ex: jogos).

Existem diversos modelos de comportamento da luz, entre eles temos:

- **Ponto de luz:** um ponto definido de onde a luz se espalha em todas as direções (ex:sol).
- **Foco de luz:** um ponto de onde a luz se propaga em formato de cone (ex: lanterna ou poste de luz).
- **Luz direcional:** um ponto de onde a luz se propaga em uma direção mas não em formato de cone e sim preenchendo toda a área.

Um **modelo de iluminação** também pode ser **global** ou **local**.

- **Modelo global:**
  No modelo global a maior parte da luz vem diretamente da fonte emissora, no entanto também é feito o cálculo da luz que reflete nos objetos, isso faz com que objetos que estejam fora do alcance da fonte de luz recebam luz de forma indireta.
- **Modelo local:**
  O modelo de iluminação local só leva em consideração a incidência de luz direta da fonte, esse modelo usa um efeito de luz "ambiente" para simular a iluminação indireta que áreas na sombra teriam. Sendo uma aproximação do modelo global o modelo local possui um custo computacional bem mais baixo.

  Exemplos de modelos de iluminação local:
- **Phong -** Reduz o modelo de luz a três componentes: **ambiente, difusa e especular**.
  A iluminação (intensidade de cor) em um determinado ponto é dada pela soma dessas componentes e os materiais refletem essas componentes de formas diferentes.
  A componente **ambiente** é a **luz de fundo** e é usada para simular a incidência de luz indireta, essa componente é igual para toda a cena e portanto é independente da posição do objeto, da luz ou do espectador sendo apenas um fator constante em cada um dos canais do RGB.
  A componente de **luz difusa** representa a iluminação que uma superfície recebe da fonte de luz e que reflete igualmente em todas as direções, sendo assim ela é independente da posição do espectador mas é dependente da posição da luz e das propriedades da superfície (normal, refletância), seu cálculo é altamente dependente do ângulo de incidência da luz.
  A componente de **luz especular** simula o efeito de reflexão da luz em superfícies brilhosas e depende da posição da luz e do observador em relação a superfície bem como as propriedades de refletância da superfície (elementos mais refletivos irão apresentar um resultado maior nessa componente).

Diferenças entre iluminação e sombreamento (**Lighting vs Shading**)

Enquanto o modelo de iluminação efetua a cálculo de luz para cada ponto da superfície de um objeto o modelo de sombreamento faz uma simplificação e calcula apenas os valores da iluminação para os vértices da superfície, para obter os valores dos pontos entre os vértices é utilizada uma interpolação.

Modelo plano/constante (Flat/Constant):

Define uma normal para cada polígono (e não para cada vértice)

- **Iluminação:** avalia a equação da luz no centro de cada polígono usando a normal associada.
- **Sombreamento:** para cada ponto do polígono é dado o valor interpolado dos vértices.
- **Implementação:** (impreciso) comentaram que seria necessário apenas mudar o tipo de um parâmetro nos shaders de “smooth” pra “flat”.

Modelo de Goraud:

Define uma normal para cada vértice.

- **Iluminação:** avalia a equação da luz em cada vértice utilizando a normal associada.
- **Sombreamento:** para cada ponto no polígono é dado o valor interpolado dos vértices (que nesse modelo foi encontrado na etapa de iluminação).
- **Implementação:** Gouraud é idêntico ao Phong, mas a cor é calculada somente nos vértices. Dessa forma, o código do cálculo de luz deve estar no vertex shader.

Modelo de Phong:

Cada vértice possui uma normal associada.

- **Iluminação:** avalia a equação da luz em cada vértice de acordo com a normal associada.
- **Sombreamento:** para cada ponto no polígono é realizada a interpolação das normais e então é avaliada a equação da luz em cada ponto usando as normais dos pontos.
- **Implementação:** Phong é o shader default usado no projeto da cadeira. Cálculos de luz no fragment shader.

Slide 9 - textura

Existem diversas formas de se aplicar a cor a um objeto na tela, por exemplo podemos guardar a informação de cor em cada vértice do objeto.
No entanto a resolução da geometria (mesh) e da cor (textura) podem não coincidir, para resolver esse problema podemos utilizar a técnica de aplicação de textura, essa técnica consiste em mapear os vértices do objeto para uma textura (imagem) que irá possuir as informações de cores.
Essa técnica pode ser vista como se fosse embrulhado um papel com as cores em torno do objeto e portanto está sujeita a sofrer distorções, uma vez que se está tentando representar as cores de um objeto na cena (3D) em uma imagem de textura (2D), existem diversas formas de fazer o mapeamento e cada uma é mais propensa a um tipo de distorção.
O processo de mapeamento dos vértices para a textura pode gerar um efeito de aliasing, para resolver esse problema é normal a utilização de um filtro linear.
Se a resolução de uma textura for muito alta para a imagem que está sendo gerada muitos texels (pixels da textura) estarão presentes em um único pixel da imagem, para prevenir esse efeito seria correto usar uma textura com uma resolução mais baixa, no entanto se nos aproximarmos essa textura simplificada iria parecer borrada. O ideal é poder escolher dinamicamente a resolução da textura dependendo das condições de visualização, para

isso se usa o mip-mapping, que consiste em estabelecer um conjunto de cópias da textura com diferentes resoluções e escolhe o mais apropriado para cada momento.

Yan (o que eu entendi de **mipmapping**):

Se criam versões da textura em resoluções menores que a original. Quando o nível de detalhe é mais baixo (modelos mais longe da câmera), são usadas as texturas reduzidas. Já em modelos próximos a câmera, se renderiza usando a textura original.
Se usasse a textura original para toda cena, os pontos distantes ficam serrilhados (aliasing), pois haveria muito detalhe em um pixel mais distante (que deveria ter menos detalhe).

Slide 11 - Anotações de Ray-Tracing

“Lançar um raio e ver onde bate”
Rasterizador é num sentido(objeto->camera) e ray-tracing é no sentido inverso(camera->objeto)

Processo de 3 partes: como gerar o raio? como descobrir onde o raio bate? como calcular a iluminação?

Quem define os raios é a resolução do near plane, um raio por pixel ou raios em cada canto.

Por que não tem Z-Buffer? porque não precisa, ele esmaga o objeto na tela e o ray-tracing já acha quem está mais perto.

Ray-Tracing Recursivo:

Espelhar os raios, um raio incide e reflete em outras direções.

Como calcular sombras? calcular a normal da intensidade da luz + oclusão.

Para limitar a quantidade de raios tu lança os raios mais prováveis, até o resultado não ser significante e não lança quando objeto nao for reflexivo ou tiver refração.

Resuminho da Aula da anti-aliasing.

Slide 12 - Anti-aliasing

Aliasing é o efeito de "serrilhamento" que pode aparecer nas bordas de objetos em uma imagem, esse efeito é resultado da discretização do espaço de cena para o espaço de tela. Existem diversas técnicas para resolver esse problema, a mais simples é o supersampling, que consiste em gerar a imagem com uma resolução maior e, em seguida, realizar um downsampling realizando uma média entre os valores dos fragmentos. Essa técnica gera ótimos resultados mas é muito ineficiente do ponto de vista computacional.
Temporal Aliasing: Conforme o tempo passa e o objeto se move, ocorre uma troca muito brusca na cor do pixel. O efeito é como se o pixel estivesse "piscando" de cor. Supersampling também resolve esse problema. Já que calculando a média das cores em um modelo maior, resulta em trocas mais suaves.

Questões exemplo de Prova 2

Questões Teóricas

Questões exame ano passado:

1) Fale sobre algoritmos de Back Face Culling, interpolação bilinear e anti-aliasing: O algoritmo de Back Face Culling é um algoritmo que tem como função determinar quais faces dos triângulos de uma objeto estão visíveis à câmera e, portanto, quais faces precisam ser desenhadas na tela – pois serão visíveis – e quais faces não precisam ser desenhadas naquele frame – por não alterarem a cena. Este algoritmo reduz o custo computacional de desenhar objetos na tela, pois elimina a necessidade de desenhar todos os triângulos de um objeto, pois é muito mais custoso desenhar do que analisar se um triângulo está com a sua face voltada para a tela ou não. Para facilitar o modo como o algoritmo detecta para qual lado está voltada a face dos triângulos, a forma como os vértices do triângulo estão armazenados no vetor de triângulos já determina qual lado está a sua face, quando os vértices estão armazenados em ordem anti-horária o triângulo está de frente para a câmera e quando estão em sentido horário, estão com a face oposta à camera – sem precisar ser desenhado.
O algoritmo de bilinear interpolação é responsável por decidir a coloração de fragmentos a partir de uma bilinear interpolação entre as cores dos três vértices que compõem o triângulo no qual o fragmento está contido. Ou seja, se um fragmento está 10% próximo do vértice A, 30% do vértice B e 60% do vértice C, a sua cor será: (0,1 * cor de A + 0,3 * cor de B + 0,6 cor de C). Dessa forma, é possível preencher o interior dos triângulos apenas sabendo a cor dos três vértices que o compõem e, também, a localização de cada fragmento dentro do triângulo.
O algoritmo de anti-aliasing é o algoritmo responsável por tentar resolver o problema de aliasing, em português conhecido como “problema do serrilhado”. Ao converter linhas dos triângulos dos

objetos de uma cena para pixels na tela, é necessário realizar uma discretização, pois a tela é composta de uma grade discreta de pixels. Neste processo, o algoritmo utilizado para decidir quais pixels serão selecionados para representar esta reta, frequentemente não conseguem expressá-la de forma reta, ou seja, a representação da reta é feita com degraus não uniformes. Para resolver este tipo de deformidade é utilizado o algoritmo de anti-aliasing, que além de selecionar os pixels que tentam representam a reta, outros pixels no em torno também são escolhidos e coloridos gradualmente (parte com a cor da reta e parte com a cor de fundo) para suavisar e reduzir a sensação de degraus existentes.

2) Fale a diferença de uma cena gerada com rasterização para ray-tracing não recursivo

Nenhuma, a cena gerada é a mesma.

3) Faça duas suzannes girarem em torno de um ponto, uma no sentido horário e outra anti-horário

4) Faça as suzannes trocarem de cor toda vez que se tocarem. Use a técnica de colisão de esferas.

5) Converta de Phong pra Gourand

https://drive.google.com/drive/folders/0B0qUjvvzujhbdWdhb0FrOWdsbE0?usp=sharing -> Phong P, Gourard G

Questão 1 (10%): Explique o problema do serrilhado (aliasing).

R: Renata* (copiada do resumo acima)

Aliasing é o efeito de "serrilhamento" que pode aparecer nas bordas de objetos em uma imagem, esse efeito é resultado da discretização do espaço de cena para o espaço de tela.

Questão 2 (15%): Diga duas vantagens e duas desvantagens do algoritmo de ray-tracing em comparação ao de rasterização.

R: Zé

Vantagens: mais realista, capaz de simular reflexão.

Desvantagens: Computacionalmente mais custoso, mais (precisa de mais informações sobre os objetos, maybe?)

Questão 3 (15%): Explique o algoritmos de z-buffer e compare com o algoritmo do pintor.

R: Renata*

No algoritmo z-buffer verifica-se para cada polígono se tem algum outro polígono que está mais a frente (no eixo z), se não houver nenhum polígono que “tapa” o polígono testado, então o polígono testado será desenhado na tela. O algoritmo do pintor difere do algoritmo z-buffer pois se houver objetos sobrepostos(entrelaçados), pelo z-buffer é possível a visualização, já no algoritmo do pintor, não. No algoritmo do pintor não se faz comparações como no z-buffer, se desenha primeiro

o que está mais ao fundo (no eixo z) e vai se desenhando novos objetos em cima dos antigos, como se fosse uma pintura.
Fontes para melhor entendimento: https://pt.wikipedia.org/wiki/Algoritmo_do_pintor e https://www.tecmundo.com.br/video-game-e-jogos/1200-z-buffering-o-que-e-e-como-funciona.htm

Questões Práticas

Link da prova inteira com os projetos (das questões 4 e 5 práticas):

https://drive.google.com/file/d/0B3bsz8pEt9rQeXNGMGtqaEZyQmc/view

bel

Questão 4 (30%): Modifique o código da pasta "Questão 4" para que o número 3, mapeado sobre o cubo, convirja de vermelho para branco continuamente em intervalos de 3 segundos.
pasta das imagens
https://drive.google.com/open?id=0B3N-YYiD6vQ6bnNwWThaa1dTaVk

Yan: https://drive.google.com/open?id=0B4a1E4ifjTOJcWVLX21WeVBwQ1U

Natalia: https://drive.google.com/drive/folders/0B7Up2NI_nEjkRF9ZaTJMTWdxWnM

Questão 5 (30%): Para comemorar o final do semestre a Suzanne vai festejar. Modifique o código da pasta "Questão 5" para que a Suzanne esteja cercada por no mínimo 3 luzes que ficam trocando de cor, de maneira suave, como numa discoteca. Cada luz deve ter uma cor diferente e um tempo de troca de cor diferente.

Yan: https://drive.google.com/open?id=0B4a1E4ifjTOJNGtPQVlCT0ZfTjg

Questão prática do ano passado(Foi uma única questão prática) :

Fazer 3 shaders, Phong, Gourad e Flat e colocar um botão pra qnd pressionar mudar entre eles.

Renata:
https://drive.google.com/drive/folders/0B0qUjvvzujhbdWdhb0FrOWdsbE0?usp=sharing -> phong P, gourard G

Yan:
https://drive.google.com/open?id=0B4a1E4ifjTOJVURPaXJBYUlUNE0

Explicação:
phong -> gourad: passa tudo do fragment pro vertex e bota só pra sair a luz q tu calculou no vertex no shader
phong -> flat: bota "flat" antes de todas variáveis de saída do vertex e antes de todas de entrada do fragment

https://www.opengl.org/discussion_boards/showthread.php/185829-Help-with-Gouraud-Phong-Shading-in-Shaders

Questões exame ano passado: by Kellerson

1) Fale sobre algoritmos de Back Face Culling, interpolação bilinear e anti-aliasing: O algoritmo de Back Face Culling é um algoritmo que tem como função determinar quais faces dos triângulos de uma objeto estão visíveis à câmera e, portanto, quais faces precisam ser desenhadas na tela – pois serão visíveis – e quais faces não precisam ser desenhadas naquele frame – por não alterarem a cena. Este algoritmo reduz o custo computacional de desenhar objetos na tela, pois elimina a necessidade de desenhar todos os triângulos de um objeto, pois é muito mais custoso desenhar do que analisar se um triângulo está com a sua face voltada para a tela ou não. Para facilitar o modo como o algoritmo detecta para qual lado está voltada a face dos triângulos, a forma como os vértices do triângulo estão armazenados no vetor de triângulos já determina qual lado está a sua face, quando os vértices estão armazenados em ordem anti-horária o triângulo está de frentepara a câmera e quando estão em sentido horário, estão com a face oposta à camera – sem precisar ser desenhado.
O algoritmo de bilinear interpolação é responsável por decidir a coloração de fragmentos a partir de uma bilinear interpolação entre as cores dos três vértices que compõem o triângulo no qual o fragmento está contido. Ou seja, se um fragmento está 10% próximo do vértice A, 30% do vértice B e 60% do vértice C, a sua cor será: (0,1 * cor de A + 0,3 * cor de B + 0,6 cor de C). Dessa forma, é possível preencher o interior dos triângulos apenas sabendo a cor dos três vértices que o compõem e, também, a localização de cada fragmento dentro do triângulo.
O algoritmo de anti-aliasing é o algoritmo responsável por tentar resolver o problema de aliasing, em português conhecido como “problema do serrilhado”. Ao converter linhas dos triângulos dos objetos de uma cena para pixels na tela, é necessário realizar uma discretização, pois a tela é

composta de uma grade discreta de pixels. Neste processo, o algoritmo utilizado para decidir quais pixels serão selecionados para representar esta reta, frequentemente não conseguem expressá-la de forma reta, ou seja, a representação da reta é feita com degraus não uniformes. Para resolver este tipo de deformidade é utilizado o algoritmo de anti-aliasing, que além de selecionar os pixels que tentam representam a reta, outros pixels no em torno também são escolhidos e coloridos gradualmente (parte com a cor da reta e parte com a cor de fundo) para suavisar e reduzir a sensação de degraus existentes.

2) Fale a diferença de uma cena gerada com rasterização para ray-tracing não recursivo

A imagem gerada utilizando a técnica de ray-tracing não recursivo faz a simulação da física da dissipação de luz pela cena, ou seja, é simulado um raio (um vetor) saindo a partir da posição da câmera e passando por cada pixel da tela de visualização até que este atinja um objeto – se houver dentro do frustum. Ao atingir o objeto, é disparado um raio em direção à(s) fonte(s) de luz, para verificar qual a iluminação global recebida neste ponto. Dessa forma, este algoritmo consegue simular de forma próxima à realidade a iluminação , com isso, a qualidade das cenas são superiores em relação à técnica de rasterização, no entanto, ray-tracing possui alto custo computacional e por isso, costuma ser utilizado na geração de cenas que não sejam de tempo real – como filmes, por exemplo.
Por outro lado, a técnica de rasterização gera suas cenas removendo a componente Z dos objetos contidos em seu frustum, ou seja, achatando os objetos de um plano 3D para 2D. A iluminação dos objetos da cena é simulada sem base na forma como a luz é fisicamente propagada pelo ambiente, e sim em técnicas de simulá-la de forma bem menos custosa. A iluminação de um objeto, muitas vezes, é feita com a soma de três tipos diferentes de iluminação, que são a luz ambiente, difusa e especular. A iluminação ambiente é uma luz “mínima” que cada objeto possui, independe da posição do objeto em relação a iluminação e é feita para simular a luz refletida de outros objetos em locais que não há iluminação direta. A iluminação difusa simula a iluminação direta dos objetos em relação à iluminação, ou seja, é verificado a ângulo entre o raio de luz e a face do objeto para determinar a intensidade de luz difusa. E a iluminação especular, simula os reflexos dos objetos, feita calculando o ângulo da visão da câmera e da iluminação (além de considerar o tipo de material usado, assim como nas demais). Com rasterização, o custo computacional de gerar cenas é muito menor, o resultado gerado é razoavelmente bom, no entanto, quando comparado ao ray-tracing, seu resultado é bastante inferior quando pensamos em simular a realidade. Apesar disso, rasterização é bastante utilizado na geração de cenas em tempo real, como em jogos, por exemplo.

3) Faça duas suzannes girarem em torno de um ponto, uma no sentido horário e outra anti-horário

4) Faça as suzannes trocarem de cor toda vez que se tocarem. Use a técnica de colisão de esferas.

Yan: https://drive.google.com/open?id=0B4a1E4ifjTOJMExCQ0hPd2h5SG8

5) Converta de Phong pra Gourand

6) Pq usamos a variável ambient no ray tracing?
R: Renata *

A variável ambient define a luz de fundo, e no ray tracing quando o raio de luz não atinge nenhum objeto colorimos o pixel baseado nessa variável

Nath: Eu acho que essa variavel é a cor default do objeto e ajuda a definir a cor de outro objeto que o reflete (ray-tracing recursivo), dai combinada com luz e tal vai dar o realismo, nao sei tbm mas eu interpretei assim.

Resumo Nicolas:

Rasterization:

Raster Display: the screen is a discrete grid of elements called pixels. Drawing on it is basically turning the pixels "on".

In computer graphics we use geometric primitives, and thus, they need to be converted to pixels in order to be drawn in the raster display. **Consequently, rasterization is basically the selection of pixels.** It is a recurrent process, so it needs to be fast and optimized.

A few important terms:

Pixel: smallest element on screen, usually rectangular or circular.

Aspect Ratio: ratio between physical and pixel dimensions.

Dynamic Range: ratio between the minimal (=!0) and maximal light intensity emitted by the displayed pixel. Measured in bits.

Resolution: number of distinguishable rows and columns on the device. Can be measured in absolute values, e.g. 1980x1080p, or in relative values, e.g. 300p per inch.

.Screen Space: discrete 2D coordinate system representing the screen pixels.

Object Space: 3D cartesian coordinate system of the objects universe.

A vertex can have a few attributes, like normal, color and texture.

The rasterization can be divided in two steps: drawing lines and filling polygons.

As of Line Drawing, there is **Bresenham's algorithm** to choose which pixels to draw. It takes each column and chooses exactly one pixel for each. It needs that the distance between X0 and X1 to be greater than the distance between Y0 and Y1, and also X0 needs to be smaller than X1 and, in addition, Y0 smaller or equal to Y1.

After selecting the pixels, we need to choose which color to use on each. This can be achieved in various ways, such as: linear interpolation for the borders and bilinear interpolation to fill the polygons.

Linear Interpolation takes the vertex attributes from the vertex that form the line and interpolate them proportionally to the line size, such as, in the middle of the line, there will be 50% of each vertex attributes.

On the other hand, **bilinear interpolation** interpolates the attributes in two directions, as columns and lines. This way it is possible to also fill the polygons.

Anti-aliasing smoothens the lines by choosing more than one pixel from each column, with pixels around the originally chosen one gradually "painted".

Illumination:

An illumination model describes **inputs,** assumptions and **outputs** used to calculate illumination (color/brightness) of surface elements. Usually it includes the following elements:

Light Attributes: intensity, color, position, direction, shape.

Object Surface Properties: color, reflectivity, transparency.

Interaction between **lights** and **objects**.

Illumination models can be physically-based or non-physically-based depending if they use real physics input data.

This real data can be used to achieve a more realistic illumination, but it is rare to have enough information to specify all inputs, so **physically-based models** require accurate input and, sometimes, need to make a few assumptions, and, consequently, they take longer to compute.

On the other hand, **non-physically-based models** just need a model that is good enough for the application and focus on speed and efficiency use of computing resources.

Global Illumination: most light striking the surface element comes directly from a light emitting source (**direct illumination**). Sometimes light from a source is blocked by another object, resulting in **shadows**. However, objects in the shadow can still receive **indirect light** from light bouncing on other objects

Local Illumination: takes only direct lighting information and is an approximation of the global illumination. Usually involves the use of an "ambient" element to simulates indirect lighting (this element is applied to all objects).

Phong Model: reduce the light model to three simples components: **ambient**, **diffuse** and **specular** light. In this model the illumination (color intensity) in a point is equal to ambient + diffuse + specular. This model allows materials to reflect each component differently, though it needs the specific material coefficients to achieve this.

Ambient Component: this component is independent of the object's position, viewer's position and light's source position. It depends solely on a constant factor (in each R, G, B channels).

Diffuse Light Component: illumination that a surface receives from a light source that reflect equally in all directions. It is independent of the viewer's position, but depends on the light's position and surface properties, such as normal and reflectance. Based on Lambert's Law, it uses the normal vector to determine the intensity of the light. It uses the following formula:

$$K * I * \cos O;$$

where K is the material diffuse reflection constant, I the light intensity and **O the angle between light vector and surface normal.**

Specular Light Component: this component represent the light reflection from shiny surfaces. The color depend on the material and a constant of how it scatter lights. For example: shiny surfaces, such as metals or a mirrors, reflect more light. The specular light depends on both light source position and view position. **It uses the the angle between the reflective ray vector and view direction vector.** If this angle is small, the viewer sees specular light. The Phong lighting model uses the following formula to calculate specular light:

$$K * I * \cos ^ f (O);$$

Where K is the material specular reflection, I the light intensity, O the angle between reflective ray and view vector and f the surface property for specular highlight.

If there are m lights, we sum each light color to get the resulting one.

Textures:

A texture is a 2D image and it needs to be mapped to a 3D model. To do so, it needs something called **Texture Mapping**, that allow the transformation from 2D to 3D coordinates. There are various mapping techniques.

Sometimes the texture is separated for different segments, e.g. a texture of a man can be separated in different section such as hands, head, torso and etc..

Encode: in this technique each triangle encode 2D coordinates (U,V). The same vertex can compose different triangles and, as result, it may have different UVs based on which triangle is being referred.

Texture Filtering: Sometimes, the texture quality may not appear to be very good. This happens because the fragment shader simply picks takes the texel that is at the (U,V) coordinates, and continues happily. We can improve this using various methods, such as:

> **Linear Filtering:** With linear filtering, texture() also looks at the other texels around, and mixes the colours according to the distance to each center. This avoids hard edges, and it's a much used method, even though it's still not the best one.
>
> **Anisotropic Filtering:** This one approximates the part of the image that is really seen through the fragment. For instance, if a texture is seen from the side, and a little bit rotated, anisotropic filtering will compute the colour contained in the texels that match this fragment format by taking a fixed number of samples (the "anisotropic level") along its main direction.

Mip-Mapping: is the use of various pre-computed results to different object sizes. As a result when an object is scaled up or down, it changes the texture and, as a result, the distortion from the interpolation is greatly diminished.

Visibility:

As always in computer graphics, the optimization is the main objective. So, in order to avoid the cost of drawing objects that are not necessary, there are some techniques that can be applied. This techniques can be techniques such as **culling**, that avoid to draw all objects that are outside of the camera frustum, or **clipping**, that clips the objects and avoid drawing the parts that are not visible.

Frustum Culling: tests if an object is inside the frustum.

> **Brute force:** needs to check each triangle of each model, what is very costly computational wise.
>
> **Bounding Box:** creates a box that contains the model inside, so it checks for the limits of the box instead of every triangle. Can be adapted into bounding sphere, cylinder or prototype.
>
> **Quadtree:** creates spatial subdivision in a hierarchical sense. Consider the whole scene a big area and in the next level divides this area in 4 equally smaller ones, and so on until a certain level is achieved. This allows to test just objects that are in the areas viewed by camera. So it cuts down a good part of the scene and after it, the bounding box comes in to finish the job.
>
> **Octatree:** uses the same concept as quadtree, but divide the areas vertically too, dividing each area in 8 areas (like dividing a cube in 8 equally smaller cubes).

Occlusion Culling: test if an object inside the frustum is occluded by other. This way it avoids drawing unnecessary objects.

Backface Culling: it test the normal of the faces to determine which face is visible by the camera and decide which face need to be drawn. For this to work the mesh needs to be built using a defined orientation, clockwise or counterclockwise.

Z-Buffer: is a fragment level algorithm that put elements in a buffer, ordered by their Z points, and then draws the element with the Z closest to the camera. This way, the occluded elements are not drawn.

Clipping: clipping a point is simple, you just check the point position. In contrast, lines are a little bit trickier, they can be inside, out, they can be clipped with one point inside and they can be clipped with none. This can be achieved using the parametric equation of the line to check if there is an intersection with any of the boundaries of the frustum.

Cohen-Sutherland Line Algorithm: this algorithm uses the divide-and-conquer rule. If it can neither trivially accept or reject a line it subdivides it in two segments and test again. Before doing this it calculates the intersections with the boundaries and divide the lines there. It uses and outcode of 6 bits to define the plane where the line is.

Ray Tracing:

A ray tracer is a mapping of rays from the camera (eye) through each pixel to the objects in the scene. Each pixel returns either the ray intersection with the nearest object or no intersection at all. To render a scene, a ray is cast from each pixel, and the returned value (color) is recorded.

The ray tracer iterates over pixels instead of vertices as in rasterization. It follows the ray that passes through each pixel to find what it needs to draw. For that, it needs to generate the rays in the world coord. The ray starts in the camera's position (P), eye point, and shoot towards a point on the film plane, creating a directional vector (d). In a parametric form, any point on the ray can be described by P + td where t is a non-negative real value. The d vector can be obtained by d = normalize(Q-P) where Q is the point on the film plane.

Recursive Ray-Tracing: each ray trace reflect on the surface in a number of N directions. This creates a recursion and allows effects like reflection, transparency and refractions.