P2 VTK

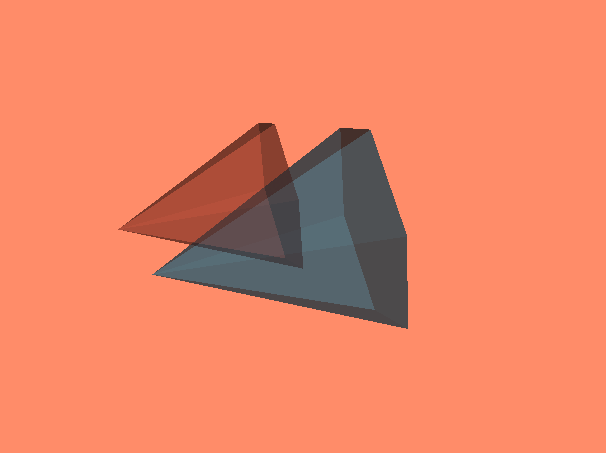
Cristiano Nicolau, 108536

Miguel Miragaia, 108317

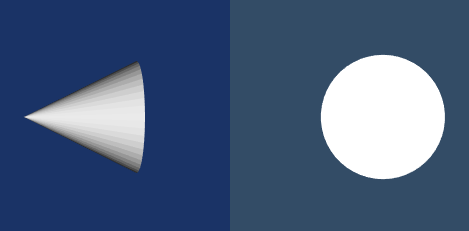
Information Visualization, 2024 University of Aveiro)

# Multiple Actors

O código inical cria um cone em uma janela de renderização. O cone é exibido com um fundo laranja e pode ser rodado interativamente na janela.

As modificações que fizemos adicionam propriedades visuais e manipulam atores. Primeiro, alteramos as propriedades do primeiro cone, alterando a cor, a difusão, etc. Em seguida, criamos um segundo cone usando um objeto vtkProperty, o que permite compartilhar as mesmas propriedades entre vários atores. O segundo ator é posicionado deslocado no eixo Y . Após isto alteramos a opacidade de ambos os cones para 0.5, o que os torna parcialmente transparentes. Assim, ao sobrepor os cones, podemos observar como os cones interagem.

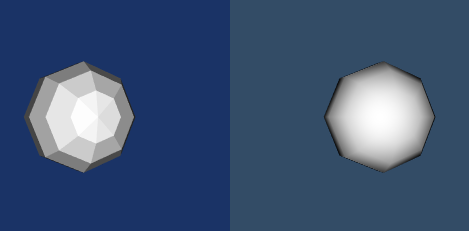
# Multiple renderers

Nos começamos por pegar no código da aula anterior e criar uma janela com dois viewports, onde ambos mostram o mesmo cone, mas com câmaras e cores de fundo diferentes. Primeiro, criamos o cone e aumentamos sua resolução para 50 para ficar mais arredondado. Depois, criamos dois renders, ren1 e ren2, que controlam as duas partes da janela: o primeiro ocupa a metade esquerda e o segundo ocupa a metade direita. Cada render tem uma cor de fundo distinta para diferenciá-los.

Posicionamos a câmara no primeiro render diretamente em frente ao cone, enquanto no segundo render, a câmara foi girada em 90 graus no eixo horizontal, para que o cone fosse mostrado de um ângulo diferente, depois usamos a animação da aula anterior de forma a fazer o cone girar nas duas metades da tela ao mesmo tempo. A janela foi configurada com tamanho 600x300, permitindo que cada rander ocupe metade da tela e mostrando a rotação do cone de ângulos diferentes.

# Shading options

Alteramos o exercício da linha anterior, para ao invés de usar cones, usar esferas. Cada esfera é como no exercício anterior, tem o seu lado e a sua cor especifica de fundo. A câmara em cada render é ajustada para que as esferas sejam vistas de diferentes ângulos, e a câmara no segundo render é rodada em 90 graus. A primeira esfera usa Flat shading, e a segunda usa Gouraud shading, permitindo comparar os efeitos das sombras. Como no exercício anterior as esferas rodam, mostrando o objeto em ângulos diferentes e no fim é possível interagir com o objeto.

1. No Flat shading, a superfície da esfera composta de planos retos, com cada face triangular da esfera a ser colorida de forma uniforme. A sombra do plano atribui uma cor única a cada face da esfera, sem fazer uma transição suave entre elas, resultando numa aparência mais angular.

2. No Gouraud shading, a superfície da esfera tem uma aparência mais suave do que a anterior. As cores são alternadas entre os vértices das faces da esfera, resultando em um gradiente de cor entre os vértices.

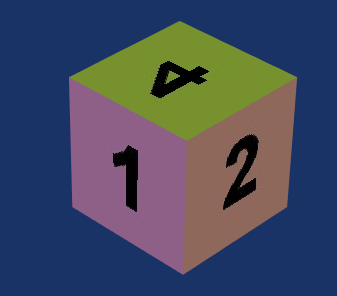
# Textures

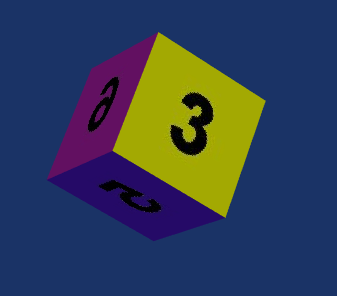


Começamos por criar um plano 2D e aplica uma textura nele, usando uma imagem JPG (lena.JPG). Primeiro, defenimos o plano com vtkPlaneSource e carregamos a imagem com vtkJPEGReader. A imagem é aplicada como textura ao plano usando. O plano é mapeado e exibido e permitimos interação de zoom e rotação, enquanto a visualização é renderizada continuamente.

Ao alterar o tamanho do plano com SetOrigin, SetPoint1 e SetPoint2, a textura também se ajusta às novas dimensões do plano. Ou seja, a textura é mapeada para cobrir a área do plano, e as mudanças no tamanho do plano afetam como essa textura é exibida.

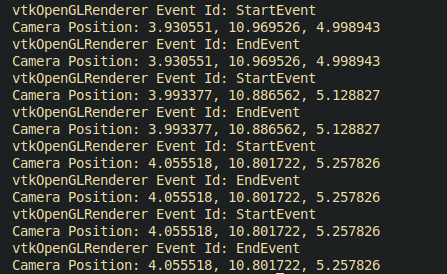
# Transformation





Começamos por criar uma função create\_textured\_plane que cria planos 2D e aplica transformações (rotação e translação) mapeando a imagem JPEG (indicada por nos, face do dado) como textura no plano. O plano é então mapeado e exibido através de um ator, que é adicionado a um render. Na main, o código define diferentes rotações, translações e texturas, e para cada conjunto de valores, chamamos a função para criar e adicionar os planos à cena, criando assim um dado. O fundo da cena é definido como uma cor específica e a câmara é posicionada para observar todos os planos. A janela de renderização é configurada com um tamanho de 800x800, e permitimos a interação com o dado de forma a ter varias visualizações do mesmo.

# CallBack



A principal mudança que fizemos em relação ao anterior foi a adição de uma callback para capturar eventos durante a renderização. Foi criada a classe vtkMyCallback, que recebe o renderizador e, quando um evento vtkCommand.StartEvent & EndEvent ocorre (no fim da renderização), imprime o nome do chamador e a posição da câmera. Essa callback foi adicionada ao renderizador com renderer.AddObserver.