Visão Computacional e Arte Digital

MAC0499 - Trabalho de Formatura Supervisionado

Aluna: Flavia Nascimento Ost
Orientador: Roberto M. Cesar Jr.

23 de março de 2008

Sumário

1 Introdução

Quando falamos em arte digital, grande parte das pessoas pensa em imagens criadas ou manipuladas no computador, animações em 2D ou 3D. No entanto, existem outras formas de arte digital, se considerarmos toda arte criada usando o computador como meio. Esses trabalhos podem ser facilmente encontrados no FILE - Festival internacional de linguagem eletrônica[1], um evento dedicado à discussão sobre e exposição de trabalhos do gênero. O tipo de trabalho exposto no FILE é muitas vezes chamado de *software art*, para evidenciar a idéia de que o software em si é o trabalho artístico, e não aquilo que pode ser gerado por ele (como é o caso de algumas produções que se adequam à área de computação computação gráfica).

Para ilustrar melhor a idéia de *software art*, podemos dar alguns exemplos de trabalhos apresentados no FILE 2007.

- Alter Ego, de Alexa Wright (Reino Unido) - O usuário tem a imagem de sua face capturada por uma câmera e depois mapeada em um modelo 3D. Por alguns segundos, a face digitalizada age como um reflexo, e logo depois passa a fazer diversas expressões faciais ou movimentos dependendo do que o usuário faz frente a câmera.

- Tetris: Estudos, de Andrei R. Thomaz (Brasil) - Variações da interface gráfica do conhecido tetris, com o objetivo de mostrar características visuais do jogo geralmente não percebidas. Na primeira variação, a grade ao fundo do jogo e a forma construída pelas peças é ressaltada; na segunda, o contraste entre peças e fundo é bastante reduzido; na terceira, as peças recebem a aparência de fundo, e o fundo recebe a aparência de peças; na última, as peças são transformadas em pontos coloridos.

- Oups!, de Marcio Ambrosio (Brasil) - O usuário fica frente a uma câmera e sua imagem é inserida num cenário com animações, onde alguns objetos podem seguir a pessoa.

- Rectable, de Interactive Sonic Systems Team (Espanha) - Movimentando objetos próprios do trabalho, podem ser gerados diversas combinações de som eletrônico.

O avator foi desenvolvido pelo aluno Marcos Paulo Moretti, aluno do IME, em colaboração com os artistas Maria Hsu e Ricardo Barreto, e apresentado no FILE 2006, sendo um trabalho relacionado a arte digital. A versão de 2006 do programa apresenta uma face tridimensional capaz de 'falar', podendo conversar com o usuário de acordo com uma personalidade. Esta personalidade é implementada através de um AIML[2] (Artificial Intelligence Markup Language - basicamente uma linguagem em que um arquivo parecido com um XML pode ser interpretado como inteligência artificial); as faces usadas são geradas pelo programa Haptek[3].

Este ano, foram feitas algumas modificações no avator, como a existência de duas personagens em vez de uma, a possibilidade de estas fazerem expressões faciais e a adição de um módulo de visão, que permite a personagem a reagir às imagens captadas por uma webcam. A implementação dessas novas características serão discutidas mais detalhadamente nas próximas seções.

2 Conceitos e Ferramentas utilizadas

No avator podemos constatar a comunicação entre alguns programas, e o uso de uma biblioteca para captura e manipulação de imagens que auxiliam na execução do algoritmo de detecção de movimento escolhido.

2.1 AIML

AIML - Artificial Inteligence Markup Language - é uma linguagem usada para representar de forma bastante simples uma inteligência artificial. Ela é semelhante a um XML, usando tags para descrever uma coleção de frases relacionadas a padrões que devem ser respondidas. As principais tags são $< AIML >$, usado para representar o início de um documento com linguagem AIML, $< CATEGORY >$, que pode marcar um grupo de conhecimentos ou assunto, $< PATTERN >$, que indica o padrão para o qual uma determinada frase será dada, e $< TEMPLATE >$, que indica a resposta em si. Para visualizar melhor, um pequeno exemplo:

$< AIML >$
...
$< CATEGORY >$
$< PATTERN >$BOM DIA$< /PATTERN >$
$< TEMPLATE >$
Bom dia!
$< /TEMPLATE >$
$< /CATEGORY >$
...
$< /AIML >$

Assim, quando o J-Alice, programa que recebe frases e retorna as respostas definidas por um AIML associado, receber um "Bom dia", a resposta retornada será "Bom dia!". É interessante citar ainda que pode-se usar expressões regulares para definir o padrão, não apenas um conjunto de palavras simples.

O arquivo AIML usado no Avator foi populado pelos artistas, usando citações de personagens escolhidas por eles.

2.2 Haptek

O programa Haptek tem funcionalidades voltadas para criação de um avatar tridimensional. É possível usá-lo para modelar faces 3D, fazer conversão de texto para fala, e sincronizar as duas ações anteriores - criar uma face que, dado um texto, ela o enuncia em voz alta.

Existem algumas possibilidades para integrar o Haptek a uma aplicação, através de wrappers para javascript ou C++, pluggins para os navegadores Mozilla, Internet Explorer ou Netscape, entre outras opções. Uma vez integrado, os comandos para o programa são passados na forma de script.

2.3 OpenCV

Uma biblioteca originalmente desenvolvida pela Intel, OpenCV[4], versão diminuta de Open Source Computer Vision Library, possui funcionalidades muito úteis para a aquisição e tratamento de imagens, facilitando a implementação de trabalhos na área de visão computacional, como segmentação de imagens, reconhecimento de faces, detecção de movimento entre outros. Possui funções para captura de imagem de câmera, execução de operações sobre essas imagens, possibilidade de facilmente exibí-las, etc. É possível encontrar também tutoriais e informações sobre algoritmos já implementados em sua página de documentação[5].

2.4 Algoritmos para Detecção de Movimento

Algumas opções de algoritmos foram discutidas com um dos membros do grupo CreatiVision, Arnaldo Lara.

2.4.1 Detecção de movimento por diferença entre imagens consecutivas

É armazenada uma imagem I fornecida pela câmera no instante t. No instante $t + \delta t$, captura-se uma imagem K da câmera, e calcula-se uma matriz M que armazena a diferença em valor absoluto, pixel a pixel, entre K e I. São determinados limiares x, para valores de elementos de M, e y ,para número de elementos de M que ultrapassem x. Se o número de elementos cujo valor ultrapassa x for maior que y, é considerado que houve movimento.

O problema desse algoritmo é que ele não detecta movimento se um usuário malicioso fizer movimentos muito lentos (a diferença entre a imagem do instante t e $t + \delta t$ é menor que o limiar) ou mesmo se um objeto for introduzido na cena e lá permanecer por mais de uma iteração do algoritmo.

2.4.2 Detecção de movimento por subtração de fundo

É feito um treinamento inicial em que guardamos um fundo padrão, uma imagem que pode ser representada por uma matriz P, sem perturbação. Esse padrão deve ser estático ou com muito pouca variação.

A partir daí, a cada intervalo de tempo δt, verificamos se houve movimento. Para isso, criamos uma matriz M que armazena o módulo da diferença (pixel a pixel) entre a

imagem A captada atualmente pela câmera e o fundo padrão. Se o número de elementos de M com valores maiores que um limiar x ultrapassar uma porcentagem y, o programa considera que há alguém na frente da câmera.

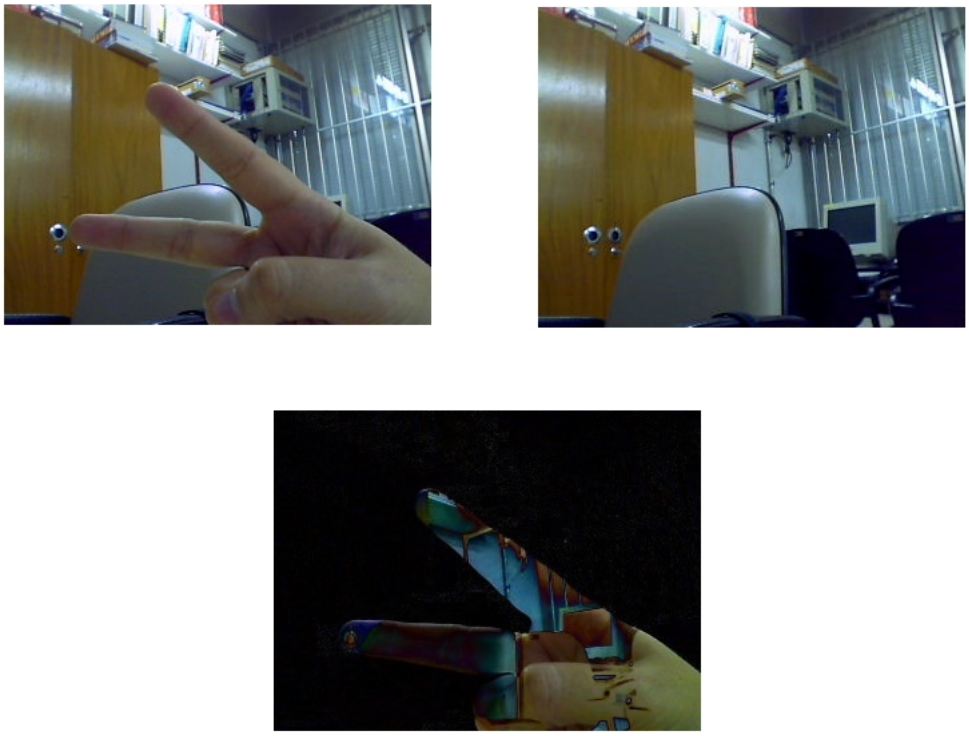

Figura 1: Exemplo de subtração de fundo. Tela atual(Matriz A), fundo padrão(Matriz P) e resultado da subtração(Matriz M).

As figuras acima são representadas com o sistema RGB. Para simplificar, digamos que cada ponto é representado por um vetor $[Red, Blue, Green]$, de forma que se esse vetor possuir valores $[0, 0, 0]$ o pixel recebe cor preta, se esse vetor possui valores $[255, 255, 255]$ ele recebe cor branca.

Quando um pixel $[i, j]$ de M tem valor $[0, 0, 0]$, quer dizer que a diferença entre o pixel $[i, j]$ da matriz atual A e o pixel $[i, j]$ da matriz padrão P é 0, ou seja, não ouve variação naquela área. Teoricamente, quando um pixel de M tem algum de seus valores Red, Blue ou Green diferente de zero significa que a imagem atual A tem diferenças em relação ao fundo padrão P, e portanto houve movimento. No entanto, pequenas variações entre P e A ocorrem naturalmente, devido a luz mudar levemente de intensidade ou mesmo a alguns algoritmos próprios da câmera, que tentam estabilizar contraste e cores da imagem

capturada. Para evitar que essas pequenas diferenças sejam consideradas um objeto que foi introduzido no ambiente, definimos o limiar x, que é o mínimo valor que Red, Blue ou Green de um pixel de M deve possuir para considerarmos que aquele pixel possui variação não natural. Além disso, criamos um limiar y que estabelece um número mínimo de pixels que devem ter essa variação não natural para acreditarmos que há um objeto na frente da tela.

3 Implementação

3.1 Avator 2006

A primeira versão desse trabalho foi feita em 2006, pelo aluno Marcos Paulo Moreti. Ela era composta por duas telas, cada uma gerada por um programa diferente.

A tela à esquerda age como cliente, recebendo informações do usuário através da caixa de texto. No programa estão implementadas algumas opções de configuração, para trocar a imagem de fundo ou a face que está sendo renderizada por exemplo, e se a frase digitada pelo usuário não for um comando especial para alterar essas configurações, ela é falada em voz alta pela personagem da esquerda ao mesmo tempo que é enviada ao J-alice. O programa aguarda a resposta do J-alice, que procura por um padrão no AIML em que a pergunta do usuário se encaixe. Se não a encontrar, retorna uma string vazia, o que não causa erro no programa, apenas faz com que a personagem não responda. Caso encontre uma resposta, envia a mesma para o programa que gerencia o campo de texto. Assim que este a recebe, a frase resposta é enviada ao programa que controla a face da direita.

A tela à direita age como servidor, aguardando que lhe sejam enviadas frases. Quando recebe uma, faz com que a sua face diga-a em voz alta.

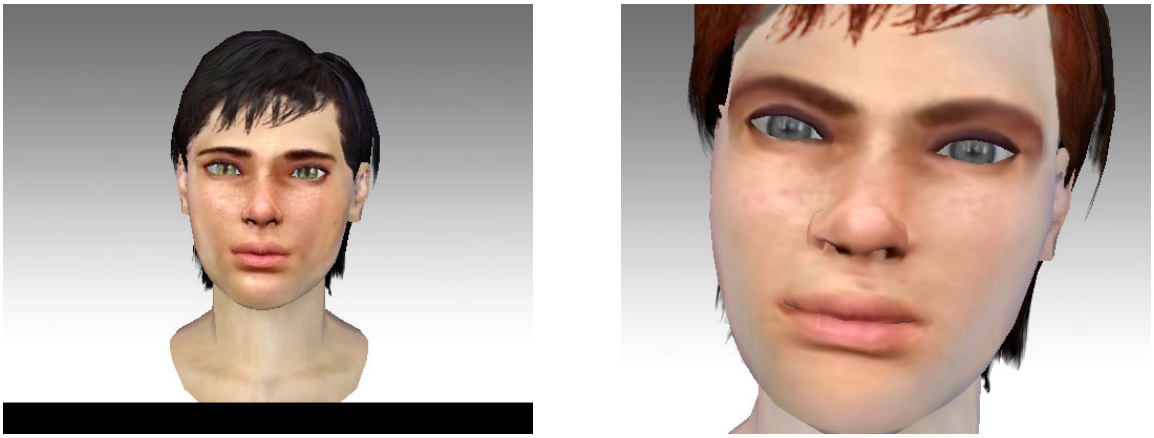

Figura 2: Telas da versão anterior do Avator.

3.2 Alterações especificadas pelos artistas

As alterações dessa seção foram feitas a pedido dos artistas envolvidos no projeto, Ricardo Barreto e Maria Hsu Rocha.

3.2.1 Mais de um personagem

Como foi descrito na seção anterior, o usuário conversava com uma única personagem. Foi sugerido que tivéssemos duas personagens, que respondessem ao usuário e trocassem mensagens entre si.

Primeiramente, foi feita uma cópia do programa responsável por gerenciar a face que responde a perguntas do usuário. Para facilitar as explicações, chamemos os programas que gerenciam as faces-personagens de Clarice e Chico (nomes das personagens cujas frases populam o AIML), e o programa responsável pelo campo de texto que recebe perguntas de Público.

Temos então dois programas com servidores, Clarice e Chico, que ficam a aguardar mensagens com frases que devem falar. Quando é inserida uma pergunta, o programa age de forma semelhante à versão original, enviando a questão para o J-alice e sua resposta para o programa que representa a personagem. No entanto, agora temos duas personagens e dois AIMLs, então há necessidade de decidir para qual a pergunta será enviada antes.

Cada vez que é recebida uma questão, escolhe-se aleatoriamente para qual dos J-alice's ativos será pedida uma resposta primeiro. Supondo que inicialmente foi escolhida Clarice, a questão é encaminhada a seu respectivo J-alice. Assim que o programa Público recebe a resposta, ela é encaminhada ao programa Clarice e imediatamente depois é enviada como pergunta para o J-alice responsável pelo AIML de Chico. Se houver uma réplica desse para a fala de Clarice, ela é enviada ao programa Chico, e logo depois é verificada a existência de uma tréplica vinda de Clarice.

Em resumo, escolhemos inicialmente uma personagem para quem mandar a pergunta do público, e depois usamos a resposta de uma personagem como pergunta para a outra. Essa troca de mensagens entre Chico e Clarice ocorre no máximo três vezes.

3.2.2 Expressões faciais

Para deixar as personagens mais interessantes, foi dada a eles a possibilidade de executar expressões faciais. O momento em que essas expressões devem ser realizadas é definido através do comando face("expressão a ser executada"), inserido em frases do AIML.

O Haptek não só permite que sejam usadas essas expressões, mas também fornece junto com sua instalação algumas definições de expressões padrão. Essas são representadas por arquivos que definem deformações, com intensidades que podem ser modificadas, em regiões da face 3D.

Assim que os programas Clarice e Chico são iniciados, carregamos as expressões que serão usadas. Quando uma mensagem é recebida, ocorre uma busca na frase pela string "face(". Se ela for encontrada, isolamos a palavra que está entre essa string e o próximo parênteses. Se essa palavra corresponder a uma das expressões definidas pelo sistema, a personagem executa a mesma enquanto diz a frase que recebeu. Por exemplo, se o

programa recebe a frase "Como vai você? face(happy)", sendo happy uma expressão definida no sistema, a respectiva expressão acompanhará a fala da mensagem.

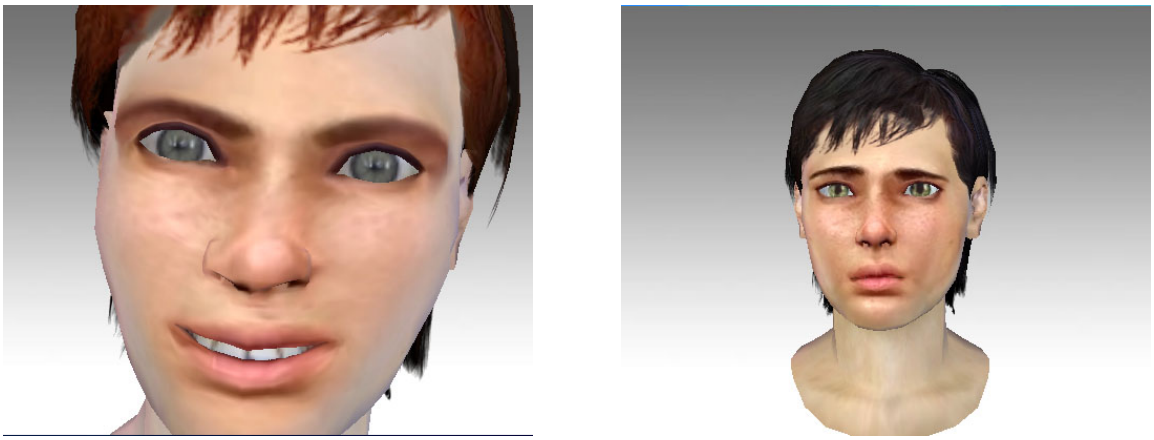

Figura 3: Exemplos de expressões faciais executadas pelas personagens: Alegria à esquerda e tristeza à direita

3.3 Módulo de Visão

O módulo de visão do avator foi concebido para que houvesse algo mais voltado para a área de visão computacional no projeto. Ele consiste em avisar ao avator sempre que houver movimento em frente a uma câmera acoplada, e o avator tem reações pré-definidas dependendo do estado do programa. Foi usada a técnica de detecção de movimento através de subtração de fundo, cuja descrição pode ser encontrada na seção 2.4.2. Os limiares escolhidos para x e y, cujo significado é explicado na já citada seção, foram $x = 20$ e $y = 0.5$.

Como foi mencionado, o avator reage à presença de movimento dependendo de seu estado atual, que é definido por uma variável booleana chamada talking, inicializada com false. As opções de ação são:

- Se não estiver conversando com ninguém (talking = false), e detectar movimento, então cumprimente o usuário.
- Se estiver conversando (talking = true) e detectar movimento, apenas aguarde uma pergunta.
- Se receber uma pergunta e não detectar movimento, questione o usuário sobre sua posição (atualmente, a pergunta emitida por um dos personagens é "Onde está você?").

- Se receber uma pergunta e detectar movimento, responda a pergunta e faça talking = true.
- Se estava conversando(talking = true) e agora não detecta mais movimento, despeça-se e abandone a conversa(talking = false).

Decidimos também que ter uma terceira face apenas para repetir em voz alta o que o usuário digitou é desnecessário, e que, a partir de agora, exibiríamos a imagem da câmera logo acima do campo de texto. A primeira maneira de exibir a imagem na tela acima do campo de texto foi através de scripts do próprio Haptek, substituindo o fundo da tela gerada por ele por imagens adquiridas da câmera. Essa opção deixou o programa bastante instável, e foi provisoriamente substituída pelo uso da função para criar janelas e mostrar imagens do próprio OpenCV.

4 Resultados

O avator é compostos por 3 programas que se comunicam, dois responsáveis por controlar faces de personagens e um responsável por receber estímulo do público (escrita ou visualização) e repassar as reações apropriadas para os outros programas. Para executá-lo, é necessário um computador com capacidade de processamento normal e gráfico razoável, já que são renderizadas faces 3D, exibidas imagens da câmera, realizadas operações entre matrizes, entre outras, ao mesmo tempo. Além disso, são necessários vários programas instalados, como o Haptek e o J-alice, e também uma câmera. Para compilar, é necessária a presença da biblioteca OpenCV no sistema, além do Visual Studio configurado para aceitá-la.

Devido a essas complicações, um video demo será disponibilizado em minha página pessoal da rede vision, http://www.vision.ime.usp.br/∼fost/video/, para facilitar a visualização do funcionamento do programa.

5 Conclusão

A visão implementada no programa é bem simples, reconhecendo apenas movimento frente à câmera através do algoritmo de subtração de fundo. Mas, assim como a possibilidade de expressar 'sentimentos' e de duas personagens artificiais conversarem entre si, essa nova característica deixou o programa mais divertido e interessante, o que é um dos principais objetivos das modificações propostas.

Futuramente, a idéia é dar ao avator a capacidade de reconhecer rostos, podendo reagir diferente se percebe um objeto ou uma pessoa.

6 Parte Subjetiva

6.1 Desafios e Frustrações

Durante este trabalho tive de aprender a lidar com várias ferramentas, bem como modificar um código criado por outro aluno.

Nunca havia trabalhado com C++, que se mostrou uma linguagem agradável - embora eu não tenha usado seus recursos a fundo, apenas criado classes novas e programado de forma razoavelmente parecida com C. Da mesma forma, tive que aprender a lidar com o Visual Studio, e descobrir como importar a biblioteca OpenCV foi um tanto quanto demorado; não que seja uma operação demasiado complexa, apenas levou algum tempo para descobrir exatamente como deveria ser feito. Ainda quanto ao Visual Studio, tive de me conformar em não entender o funcionamento de certas partes do programa, que foram geradas automaticamente pelo programa quando o Marcos começou o projeto. O fato de o programa "facilitar"gerando classes pode ser útil, mas não conseguir entendê-las é um tanto frustrante. Além disso, vários contratempos aconteceram em conseqüência da escolha de usar o Visual C++ 6 para desenvolver o projeto: Quando o computador usado originalmente para trabalhar no avator começou a apresentar problemas, descobri incompatibilidades entre algumas bibliotecas usadas por aplicações geradas pelo Visual C++ 6 com o Visual Studio 2005 e 2008, bem como com o Windows Vista, o que atrasou consideravelmente a entrega do trabalho.

6.2 Disciplinas relevantes

As disciplinas que mais influenciaram o projeto foram obviamente aquelas relacionadas a visão computacional, uma área da computação razoavelmente ativa.

Quando começamos a escolher optativas logo me interessei por computação gráfica (MAC0420), depois passei por Análise de formas (MAC0447) e Visão e processamento de imagens (MAC0417), que deram as bases e interesse necessários para começar a trabalhar no avator. Outras matérias relevantes, mas não tão específicas à área de visão, que podem ser citadas são Princípios de desenvolvimento de algoritmos (MAC0122), que forneceu os fundamentos para técnicas de programação, e Laboratório de Programação I (MAC0211) e II (MAC0242), que propuseram o aprendizado de ferramentas diversas.

6.3 Futuro

O programa ainda precisa de alguns ajustes, como encontrar um meio mais eficaz de exibir a imagem do usuário enquanto ele conversa com o avator. Além disso, por se tratar de um projeto ligado a arte digital, sempre podem haver novas modificações a medida em que forem surgindo idéias (dos artistas ou de programadores).

Creio que visão computacional não só é interessante mas também extremamente importante. Durante o desenvolvimento do Trabalho de conclusão de curso, tive a oportunidade de ver trabalhos[6] de vários alunos da pós-graduação que me deixaram impressionada. Apesar disso, e de pretender continuar desenvolvendo este projeto por mais algum tempo, percebi que essa não é a área da computação que desejo seguir profissionalmente. Até agora não encontrei uma vertente da computação com a qual consiga me imaginar trabalhando pelos próximos anos, o que causa certo desânimo, mas ainda não desisti de procurá-la.

7 Referências

[1] FILE - http://www.file.org.br/
[2] A.L.I.C.E. and AIML - http://www.alicebot.org/
[3] Haptek - http://www.haptek.com/
[4] OpenCV official page - http://www.intel.com/technology/computing/opencv/index.htm
[5] OpenCV Library Wiki - http://opencvlibrary.sourceforge.net/
[6] Creative Vision Research Group - http://www.vision.ime.usp.br/∼creativision/