Atividade de laboratório 1

P.A.E. Diego Cintra e Fábio Felix
diegocintra@usp.br, f_diasfabio@usp.br

21 de março 2018

As atividades descritas a seguir devem seguir as seguintes restrições:

- Todas devem ser implementadas **individualmente**;
- As atividades devem ser implementadas utilizando a API da OpenGL, sendo as bibliotecas `gl`, `glu`, `glut` e `glew` as únicas que podem ser utilizadas.
- As linguagens permitidas são **C** e **C++**.
- Para submissão, aqueles que optarem por utilizar Windows devem compactar todo o código-fonte como um arquivo ".zip", incluindo executável. Os que optarem por sistemas operacionais baseados em UNIX também devem enviar todo o código-fonte compactado, acompanhado de um `Makefile`.

Atividade 1

Pacotes gráficos aproximam primitivas matemáticas, descritas em termos de vértices no plano Cartesiano, de conjuntos de pixels com intensidades apropriadas de tons de cinza ou cores. Os algoritmos utilizados para desenhar retas computam as coordenadas dos pixels que ficam sobre ou aproximadamente sobre uma reta ideal. É necessário que esses pixels fiquem o mais próximo possível da reta que se quer desenhar.

A estratégia mais simples para construir uma reta é descrita pelo algoritmo *Digital Differential Analyzer* (DDA). Nesse algoritmo é utilizada simplesmente a equação da reta **incrementando/decrementando** o valor de x por 1 e obtendo o valor de y pela equação

$$y_i = m * x_i + b$$

O valor de x deve ser incrementado quando $\Delta x > 0$ e decrementado quando essa variação for negativa. Perceba que o valor de $m = \Delta y/\Delta x$ é obtido pelas variações do ponto inicial e final da reta que se quer desenhar. Agora só resta encontrar o valor de b. No caso em questão, o próximo valor de x sempre é um **incremento/decremento** do valor anterior, então $x_{i+1} = x_i \pm 1$, sendo o próximo valor de y obtido por

$$y_{i+1} = m * x_{i+1} + b \Rightarrow y_{i+1} = m * (x_i + 1) + b \Rightarrow y_{i+1} = m * x_i + b + m \Rightarrow y_{i+1} = y_i + m$$

Assim sendo, o valor de b não precisa ser definido e nossos cálculos se resumem a

$$x_{i+1} = x_i \pm 1$$
$$y_{i+1} = y_i + m$$

Podemos seguir o mesmo princípio **incrementando/decrementando** y de 1 em 1 e obtendo os valores de x com as devidas manipulações da equação da reta. Dessa maneira obtemos

$$y_{i+1} = y_i \pm 1$$
$$x_{i+1} = x_i + \frac{1}{m}$$

Para definir qual das duas abordagens utilizar, basta avaliar qual das variações é maior. **A comparação deve ser realizada com o valor absoluto das variações**. Se **o valor absoluto da** variação de x for maior, ele deverá ser **incrementado/decrementado** por 1, se não, **incrementar/decrementar** o y de 1.

Questões propostas

1. A partir do exposto acima, implemente o algoritmo DDA que recebe dois pontos e traça uma reta entre eles. A primitiva *glBegin(GL_POINTS)* deve ser utilizada para desenhar cada ponto gerado.

2. Utilizando a implementação do DDA crie funções capazes de desenhar triângulos, quadrados, círculos e elipses.

    - O círculo e a elipse são conjuntos de retas (no mínimo 30). As coordenadas inicial e final das retas podem ser calculadas com base nas Coordenadas Polares

    $$x = x_c + r_x * cos(\theta)$$
    $$y = y_c + r_y * sen(\theta)$$

    sendo que (x_c, y_c) é o ponto central e r_x e r_y são os respectivos raios de cada eixo. Para um círculo esses raios são iguais e para uma elipse, eles são diferentes. Os valores de ângulos podem ser obtidos por $\theta = 2 * \pi * i/qt_retas$.

3. Implemente as transformações geométricas translação, escala e rotação sobre os pontos dos objetos criados.

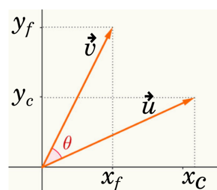
    - Depois que um objeto for desenhando, se clicar em qualquer parte da tela o objeto deverá ser transladado para esse ponto.
    - Quando as teclas + e - forem acionadas, o objeto deverá aumentar ou diminuir, respectivamente.
    - Quando as setas direcionais *direita* e *esquerda* forem acionadas o objeto deverá ser rotacionado nos sentidos horário e anti-horário, respectivamente.
    - Lembre-se de utilizar as funções *glutMouseFunc* e *glutKeyboardFunc* para manipular o evento de clique do *mouse* e as teclas que forem pressionadas no teclado.

4. Implemente um *menu* onde o usuário possa escolher qual forma desenhar. Reta, triângulo, quadrado, círculo ou elipse.

    - **Para retas**. Ao clicar o primeira vez o ponto inicial deverá ser salvo. Enquanto o usuário arrastar o *mouse* salvar o ponto que ele está como o final da reta, até que o usuário pare de arrastar o *mouse*.
    - **Para triângulos**. Pode-se utilizar o mesmo princípio das retas para os pontos inicial e final e, além disso, calcular um terceiro ponto. Para calcular as coordenadas desse ponto, podem ser utilizadas as idéias de: ponto médio da reta e altura do triângulo equilátero $h = l * \sqrt{3}/2$.
    - **Para quadrados**. Ao clicar o primeira vez o ponto inicial deverá ser salvo. Considerar esse ponto como o canto superior esquerdo do quadrado. Enquanto o usuário arrastar o *mouse* salvar ponto que ele está como o final do quadrado, até que o usuário pare de arrastar o *mouse*. Esse ponto final deve ser considerado como o canto inferior direito do quadrado.
    - **Para círculos**. O primeiro ponto clicado pelo usuário é o ponto central do círculo. Conforme o usuário arrastar o *mouse* deve ser calculado o raio do círculo. O raio é a distância Euclidiana entre o ponto central e o ponto final observado.

    $$raio = \sqrt{(x_f - x_c)^2 + (y_f - y_c)^2}$$

    - **Para elipses**. Segue o mesmo princípio do círculo, mas precisam ser encontrados raios diferentes nos eixos x e y. Os raios podem ser calculados utilizando as equações polares da **Questão 2**. O *seno* e *cosseno* do ângulo podem ser calculados a partir da formulação do produto escalar entre vetores, como apresentado nas próximas figura e equações.



$$cos(\theta) = \frac{< \vec{u} \cdot \vec{v} >}{\|\vec{u}\| * \|\vec{v}\|}$$

$$< \vec{u} \cdot \vec{v} >= x_c * x_f + y_c * y_f$$

$$\|\vec{u}\| = \sqrt{x_c^2 + y_f^2}$$

- Lembre-se de utilizar as funções *glutMouseFunc* e *glutMotionFunc* para manipular os eventos do *mouse* de clicar e arrastar.

Atividade 2

Gráficos de pizza são comumente utilizados para representação de dados simples, indicando valores presentes em cada uma das tuplas (ou colunas) de seu conjunto de dados. Essa representação gráfica já está incorporada em editores como *LibreOffice Writer* e *LibreOffice Calc*, bem como bibliotecas que facilitam a representação de conjuntos de dados, como *D3.js*. A Figura 1 demonstra essa representação, utilizando como de dados a Tabela 1.

Tabela 1: Conjunto de dados com porcentagens de utilização de linguagens de programação.

Linguagem	%
Java	26, 7
Python	20, 69
Javascript	8.53
PHP	8.33
C#	7.99
C	6.42
R	4.23
Objective-C	3.81
Swift	3.78
Matlab	2.39
Ruby	1.72
TypeScript (Ts)	1.52
VBA	1.42
Visual Basic (VB)	1.26
Scala	1.21

Questão proposta

1. Gere uma representação de um gráfico de pizza utilizando algum conjunto de dados. Um conjunto, de nome "data.csv", está incluso a essa atividade como exemplo. Lembre que a representação consiste basicamente em converter da unidade de porcentagem para graus. Cada "fatia" da pizza deve ser preenchida com uma cor distinta.

   - Seu programa deve ler o arquivo de entrada como parâmetro da linha de comando.
   - Para o desenho do círculo, pode-se utilizar as funções da atividade anterior (desenho de círculo e elipse).
   - Para facilitar a interpretação do gráfico de pizza, utilize como cor de fundo da cena a cor branca, através do comando `glClearColor(1.0, 1.0, 1.0, 1.0)`.
   - Assim como na Figura 1, permita a visualização do rótulo a que o dado pertence. Essa representação pode ser feita de duas maneiras distintas:
     - Definindo os rótulos sobre o próprio grafo de pizza, como na Figura 1;
     - Criando uma legenda com as cores e seus respectivos rótulos.

   Utilize a função `glutBitmapCharacter(void *font, int character)` para incluir texto na cena.

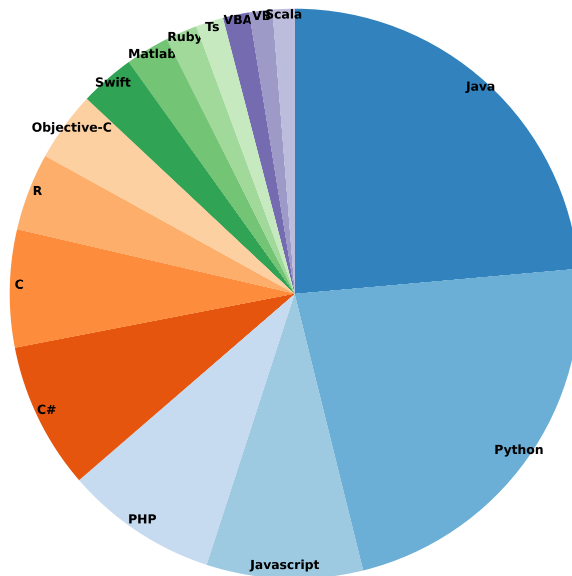


Figura 1: Representação em gráfico de pizza dos dados da Tabela 1, gerado utilizando *D3.js*.

Atividade 3

Pong é provavelmente um dos jogos de video game mais jogados e icônicos de que se tem conhecimento, se tornando bastante popular durante a década de 1970. O jogo basicamente consiste em dois jogadores, representados por retângulos verticais, dispostos nos extremos de uma cena, como na Figura 2. Uma bola é lançada do centro da cena, com direção aleatória, e o objetivo de cada jogador é defender seu "campo", esperando o momento em que o adversário não conseguirá rebater a bola, situação essa em que o jogador ganha um ponto. Na descrição original do manual de Pong [1], o ganhador é aquele que conseguir somar 15 pontos.

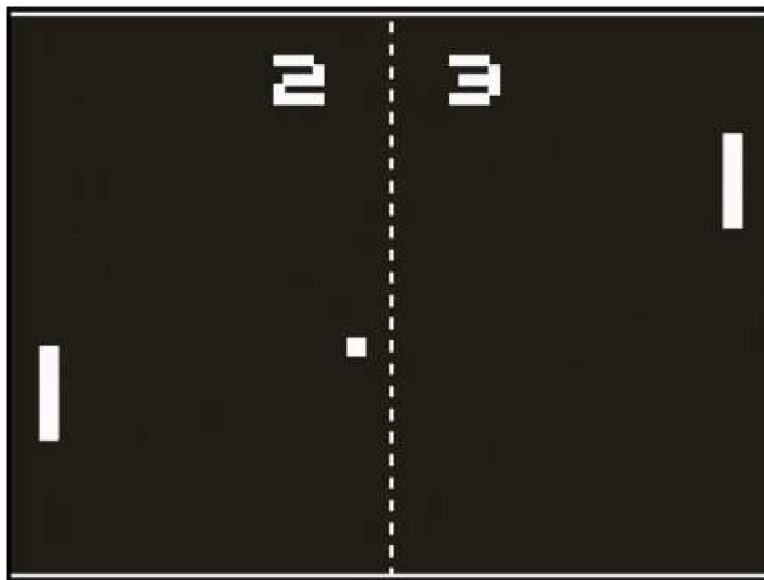


Figura 2: Imagem do jogo Pong. Obtida de `https://www.youtube.com/watch?v=ncB0ov5hT48`.

Sendo Pong um jogo com mecânica simples, inúmeras variações podem ser propostas visando incrementar sua jogabilidade. Aqui, propõe-se uma variação denominada "OpenPongL", aonde algumas novas regras são adicionadas:

- A bola contém um fator de velocidade, adicionado a cada rebate feito pelo jogador, ou seja, a cada vez que a bola é rebatida, sua velocidade aumenta em um pequeno fator ε;

[1] `https://www.robert-matthees.de/ux/product-design/pong-owners-manual-1976.pdf`

- O *rally* é uma característica do jogo que define um valor fixo R, indicando a quantidade máxima de rebates permitidos. Quando esse valor é ultrapassado, uma nova bola é inserida no jogo (sendo que a bola anterior continua em jogo), com fator de velocidade 1. O *rally* continuará ocorrendo enquanto os jogadores conseguirem rebater as bolas presentes em campo ou até haverem uma quantidade N de bolas em campo.
- Caso não haja nenhuma bola em campo, uma bola com fator de velocidade 1 é novamente inserida no meio do campo, com direção aleatória, e as mesmas regras continuam valendo.
- O jogo acaba quando um dos lados somar 10 pontos.

Questão proposta

1. Implemente o "OpenPongL" com as regras estabelecidas acima.
   - Para os controles do jogador mais à esquerda, devem ser configurados os botões "W" e "S", indicando as ações de subida e descida, respectivamente. Já para o jogador mais à direita, devem ser configuradas as setas direcionais para cima e para baixo, com as mesmas ações, respectivamente.
   - Para facilitar o entendimento do jogo, opte pela cor de fundo preta (com o comando `glClearColor(0.0, 0.0, 0.0, 1.0)`) e pelas cores dos jogadores e bolas brancas.
   - Não há necessidade de aproximar formas mais complexas do que quadrados, círculos ou retângulos.
   - Assim como visto na Figura 2, um placar deve ser disposto na parte superior do campo, novamente utilizando o comando `glutBitmapCharacter(void *font, int character)` e `glRasterPos2s(GLshort x, GLshort y)` para posicionar o texto na tela.
   - Deve-se tratar a colisão das bolas com os retângulos correspondentes aos jogadores.
   - Cumpridas as regras de finalização do jogo, pode-se simplesmente optar por reiniciá-lo, ou terminar o programa.