**SIMPLE\_PAINTER**

A canvas is a two-dimensional surface for drawing. It is represented as a two-dimensional array of colors. Each element of the canvas is a pixel. We can use an object of the class vector3 to represent a pixel color. A color has three components: red, green, and blue. Their values are inside [0, 1]. For example:

vector3 color;

color.r : red

color.g : green

color.b : blue

The painter has a current color. Use the current color to draw.

**Key usage**

‘n’, ‘N’: decrease the brush size

‘m’, ‘M’: increase the brush size

‘1’: set the brush color to red

‘2’: set the brush color to green

‘3’: set the brush color to blue

/\*

Randomly generate the colors of the bitmap.

\*/

void SIMPLE\_PAINTER::reset( )

/\*

Get the canvas dimension.

nx : number of columns; ny : number of rows

\*/

void SIMPLE\_PAINTER::getCanvasDimension( int &nx, int &ny ) const

int SIMPLE\_PAINTER::computeCanvasIndex( int x, int y, int nx, int ny )

/\*

Get the color at pixel (x,y).

Store it to color.

\*/

void SIMPLE\_PAINTER::getColorAtPixel( int x, int y, vector3 &color ) const

/\*

Show system title.

Show key usage.

\*/

void SIMPLE\_PAINTER::askForInput( )

{

cout << "SIMPLE\_PAINTER::askForInput( )" << endl;

cout << "Please use keyboard and mouse to control" << endl;

cout << "1: red color" << endl;

cout << "2: green color" << endl;

cout << "3: blue color" << endl;

cout << "n, N: decrease brush size" << endl;

cout << "m, M: increase brush size" << endl;

}

/\*

Return true if the key event is handled.

Return false otherwise.

Make sure that the brush size is not larger than mMaxBrushSize.

\*/

bool SIMPLE\_PAINTER::handleKeyPressedEvent( int key )

/\*

Return the brush size:

Return mBrushSize

\*/

int SIMPLE\_PAINTER::getBrushSize( ) const

/\*

Set the current color:

\*/

void SIMPLE\_PAINTER::setColor( const vector3 &color )

/\*

Set transparency:

\*/

void SIMPLE\_PAINTER::setTransparency( float v )

/\*

Use the brush to draw. The center is at (x,y). A disk is defined based on position (x,y) and

brush radius.

The radius of the disk is set to ~2\*mBrushSize.

Algorithm:

For each pixel in the disk centered at (x,y)

do

compute color and transparency

draw at the pixel

\*/

void SIMPLE\_PAINTER::clickAt(double x, double y)

{

vector3 color = mColor;

int ref\_s = mBrushSize/2;

double S2 = ref\_s\*ref\_s;

int s = ref\_s \* 4;

for ( int j = -s; j <= s; ++j ) {

for ( int i = -s; i <= s; ++i ) {

double rx = i;

double ry = j;

double w = 1.0; // weight

double R2 = rx\*rx + ry\*ry;

if ( R2 > 4.0\*S2 ) continue; // this pixel is too far. ignore it

if ( R2 > S2/2.0 ) { // pixel is a bit far but not too far

double k = S2/2.0 - R2;

double d = 0.003; // magic number

w = pow(2.718281828, d\*k ); // use the guassian function to compute weight w

} else {

w = 1.0; // pixel is near to the center (x,y). Set weight to a high value; here it's one.

}

//

drawAt(x+i, y+j, w\*color, w\*mTransparency);

}

}

}

/\*

Draw one pixel at (x,y) with color and weight w. If (x,y) is not inside the canvas, return.

Make sure that each color component is inside [0,1].

new color = current\_color\*(1.0-w\*w) + w\*w\*color;

\*/

void SIMPLE\_PAINTER::drawAt(int x, int y, const vector3 &color, double w )