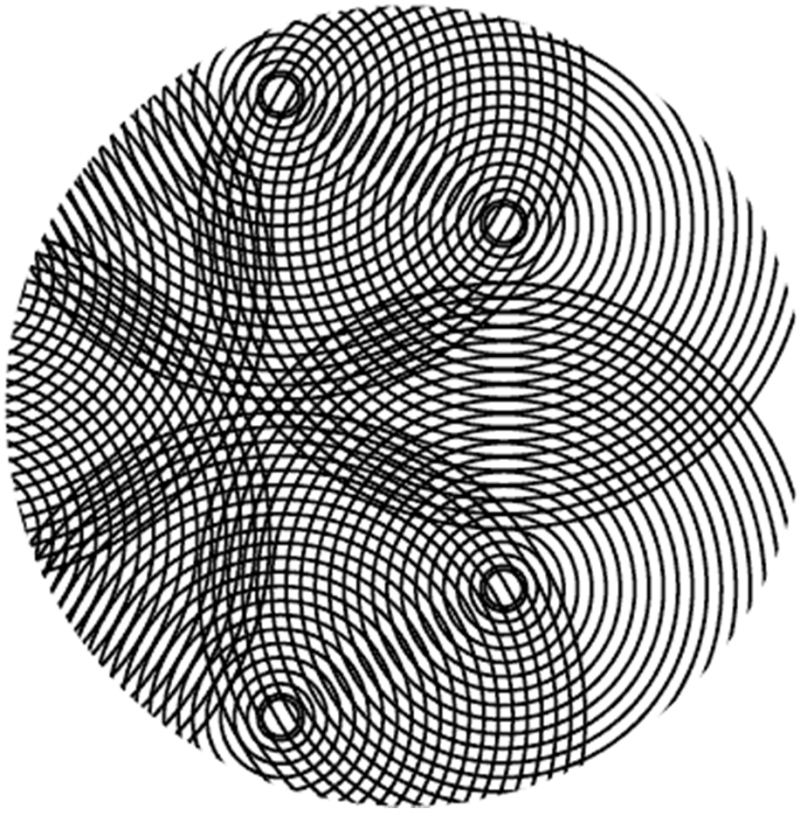


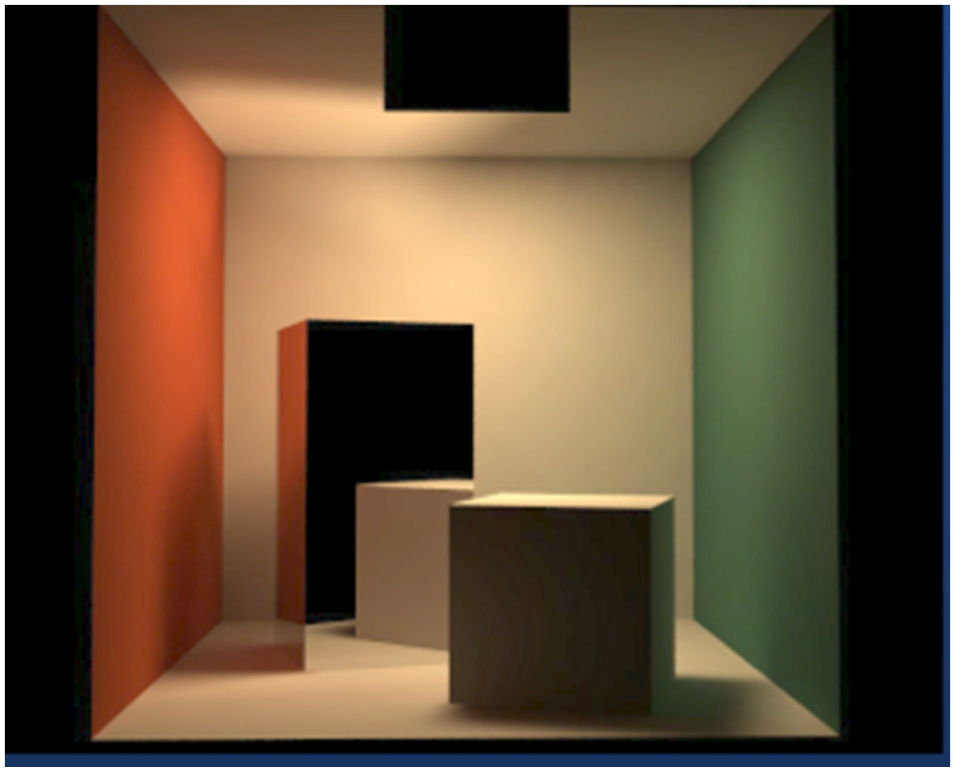
Example Image with Moire Effects
- http://www.artlandia.com/products/SymmetryWorks/moire/moire2.html

Honors 208W
Fall 2011

Programming in C#:
Generating Circles and Random Noise

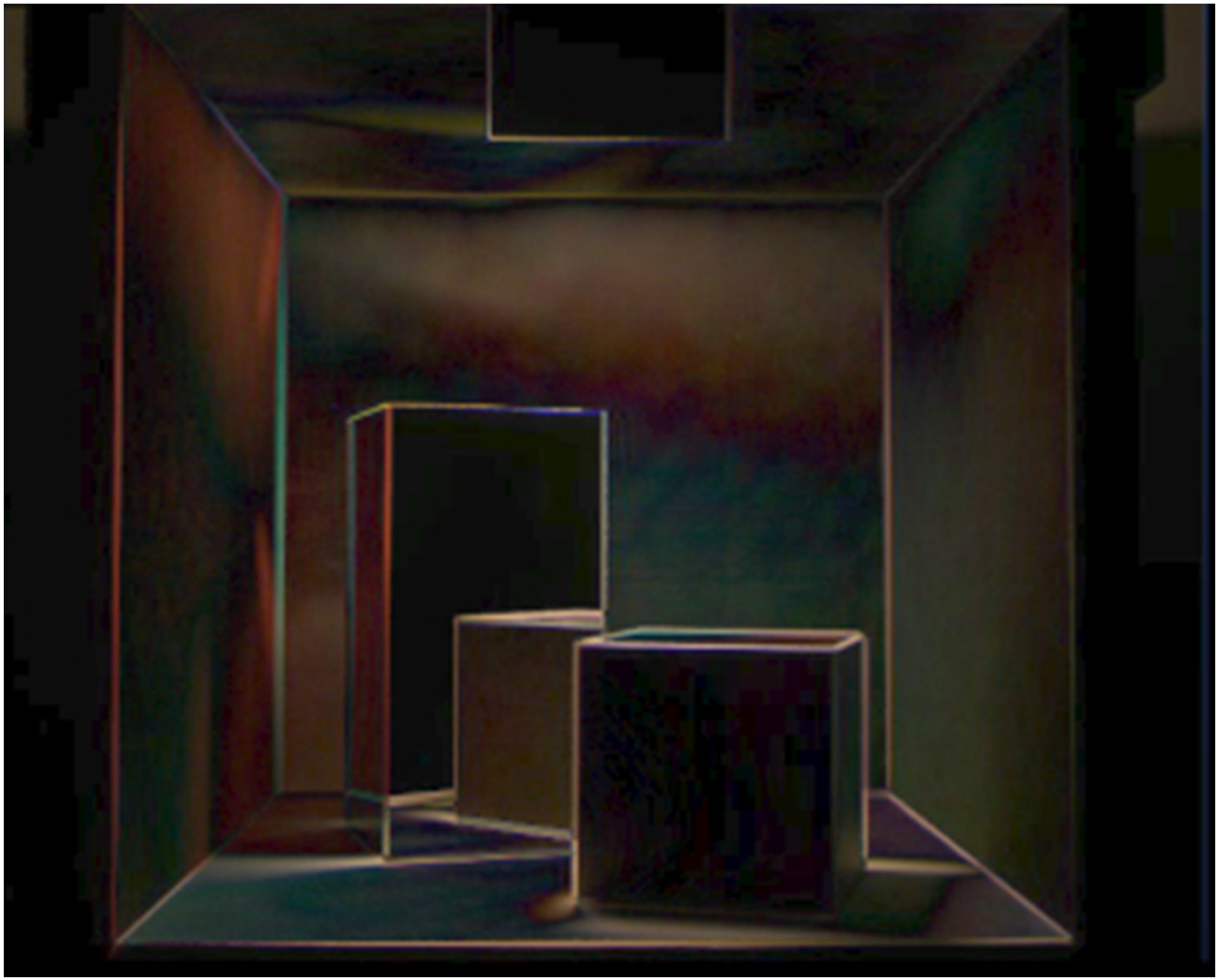
Computer Generated Images (I)

Which is the “real” image?



Computer Generated Images (II)

There are some differences…



What is C#?

C# is a **programming language**. There are many different programming languages.

Some programming languages are typically used for a very specific domain:

- JavaScript generally for web pages
- SQL for databases

Other programming languages are more general purpose:

- C++
- Java
- C#
- Visual Basic

C# is an example of an Object Oriented language.

What is Visual C#?

Visual C# is an example of an **I**ntegrated **D**evelopment **E**nvironment, this one specifically for writing C# programs.

- Graphical User Interface
- Graphical Layout of Components
- Helpful Menus to perform Common Tasks
- IntelliSense “code completion” assistance

Visual Studio is a collection of IDEs for a few different languages.

IDEs exist for other languages (both programming like Java and markup like HTML).

Getting Started in Visual C#

Start Visual Studio.

There are several different programming languages in which you can design projects and platforms for which you can design them.

- General Windows applications (eg: Windows XP, Vista, 7)
- Smart Devices (ie: devices running an embedded operating system, such as PocketPCs and Windows Smartphones)
- Web Services (typically for use within web sites)
- Text-based command windows

We will select to create a **Visual C#** project for **Windows** which is a regular **Windows Forms Application** and name it **ConcentricCircles**.

Adding an Object to a Form

There are two ways to add a new object to a form:

- Adding code manually stating to create an object of a certain type and where to place it on the form.
- Selecting an object from the **Toolbox** and then positioning it on the form.

Let's add a **PictureBox** object to our form. You can either:

- double-click **PictureBox** to have one added and automatically positioned on the form.
- single-click **PictureBox** and then drag a rectangle representing where we want it to appear on the form.

Formatting an Object on a Form (I)

After adding an object to a form, you can alter its…

- Size, Position, Color
- many more… depending on the type of object

For some of these **properties**, we can set a value either with direct manipulation (eg: drag it to where we want it) or by explicitly assigning values to certain properties of the object.

- eg: Size, Position

For others, we can *only* set the values by using the **Properties** pane or by setting values using C# code.

- eg: BackColor, Opacity

Formatting an Object on a Form (II)

Let's use the **Properties** pane (right click in the form and select Properties to see this) to make the **Size** of the **Form** itself 800 pixels wide and 600 pixels tall.

Click on the **PictureBox** shown on the form and (again) using the **Properties** pane, make the **Size** of the **PictureBox** 450 pixels wide and 450 pixels tall, and set it to have a white background (**BackColor**).

Finally, using the **Format** menu on the toolbar, let's center the **PictureBox** on the form (both vertically and horizontally).

Attaching Instructions to an **Event** (I)

There are certain **events** that a program can "listen" for on a computer such as…

- mouse movement
- key presses
- a form loading
- an object being clicked
- more...

We can indicate that when a certain event happens "live" that specific code we provide is executed.

Attaching Instructions to an **Event** (II)

In the design view for a form, if you click on the form or on an object within the form, the **Properties** pane can also be used to interact with **event handlers**.

- You can click the **lightening bolt** icon in the **Properties** pane to see a list of supported events for the currently selected visual element.

If you double-click on the name of an **event**, an **event handler** is created, and we switch to that handler in a coding window for the form.

Let's add some code saying that when the form is loaded, we will draw a circle on the **PictureBox**…

Drawing a Circle in a Picture Box (I)

Step One

Create a drawing surface in the PictureBox.

- Select the main **Form**, go to the list of **Events** and double-click the **Load** event.
- In the form's **Load** event handler (it will be named something like Form1_Load) we have to provide an **Image** object to the **PictureBox** on which we will draw, so place the following line of code inside that event handler:

```
pictureBox1.Image = new Bitmap(pictureBox1.Width, pictureBox1.Height);
```

Drawing a Circle in a Picture Box (II)

Step Two

Draw the circle.

- To start simple, in the form's **Load** event handler, we can give instructions to draw a circle on the **Image** object that we just created.
- We want to create a graphics object associated with the Image object we created.

```
Graphics g = Graphics.FromImage(pictureBox1.Image);
```

- We want to draw an ellipse via that graphics object.

```
g.DrawEllipse(Pens.Black, 0,0, 100,100);
  //Starting at coordinate (0,0) which is the upper-left corner, draw
  //   an ellipse that fits within a rectangle that is 100x100 pixels
```

Adding Buttons (I)

Let's add two buttons to our form; one to ***draw*** a circle, and one to ***clear*** the box.

Step One

Go back to the **Design** tab for the form and using the toolbox again, add two buttons to the form.

Move them to locations that you find visually pleasing.

For each, go to the Properties pane and change its **Text** property to be a word to describe what the button will do (perhaps "Draw" and "Clear").

We can also change the internal **Name** property of the button objects to be more descriptive as well if we'd like.

For each button, using the events list, create an event handler for **Click**.

Adding Buttons (II)

Step Two

We can now add C# code to the **Click** event for each of the buttons to determine what happens when the user actually clicks the button when the program is running.

- Go to the code tab of the form to add code to the event handlers.
- Move the two lines for creating a graphics object and then drawing to it from the **Load** event handler to the **Click** event handler for the "Draw" button.
- Next, we need to add a new line of code after the instructions to draw the circle which will refresh the screen:

```
Refresh();
```

- Finally, we can add some code to the **Click** event handler for the "Clear" button to clear the image:

```
Graphics g = Graphics.FromImage(pictureBox1.Image);
g.Clear(Color.White);
Refresh();
```

Iteration (ie: Loops)

Rather than just drawing a single circle on the **PictureBox**, let's draw several concentric circles.

First, we need to work out an algorithm to do this.

Then, we need to implement it!

The way to draw a circle is to call the DrawEllipse method and provide it with a color, an upper-left coordinate, and a width/height.

Let's work some ideas out on the board…

Concentric Circles

In the **Draw** button's **Click** event handler, let's replace the single call to **DrawEllipse** with the following code which calls it 15 different times, each time having the upper-left corner change as well as the size.

```
int avail = Math.Min(pictureBox1.Width, pictureBox1.Height);
for (int size=1; size<(int)(avail/2.0); size+=15)
{
   g.DrawEllipse(Pens.Black, (avail/2)-size, (avail/2)-size, size*2, size*2);
}
```

To change how many circles appear, you could just change the **15** to a different number.

New Project: Draw Random Noise

Let's create a new C# project and name it **Noise**.

In this project, we will have:

- a **Form** that is 400 pixels wide and 300 pixels tall
- a **PictureBox** that is 300 pixels wide and 150 pixels tall with a black background
- a **Button** showing the text "Create Image"

When we click on the button, an image will be generated by assigning each pixel within the **Image** object a random color.

Generating the Random Image (I)

In the button's **Click** event handler we need to create a new random number generator, iterate through every pixel position in the Image object, and assign a random color to it.

- Create a **Random** Number Generator:

```
Random rnd = new Random();
```

- Create a fresh **Bitmap**:

```
Bitmap newBMP = new Bitmap(pictureBox1.Width, pictureBox1.Height);
```

- Declare some variables for later use:

```
int red, green, blue;
```

Generating the Random Image (II)

Iterate through every pixel:

```
for (int x=0; x<pictureBox1.Width; x++) {
  for (int y=0; y<pictureBox1.Height; y++) {
        //Code to alter pixel (x,y) would go here
  }
}
```

Generate the random color channel by channel:

```
red = rnd.Next(255);
green = rnd.Next(255);
blue = rnd.Next(255);
```

Assign that color to the pixel

```
newBMP.SetPixel(x,y,Color.FromArgb(red, green, blue));
```

After the loops, assign this new image to the **PictureBox**, and refresh the screen.

```
pictureBox1.Image = newBMP;
Refresh();
```

Showing progress…

There are many ways to show progress on-screen as a long task is executing.

One way is to add a **ProgressBar** object to the form.

- Set its **Minimum** property to 0.

```
progressBar1.Minimum = 0;
```

- Set its **Maximum** property to the value to which you will count (eg: the number of times the main loop iterates).

```
progressBar1.Maximum = pictureBox1.Width;
```

- Increment its **Value** property within the loop.

```
progressBar1.Value = x;
```

Saving an image to a file…

The simplest way to start is with a command like:

```
pictureBox1.Image.Save("test.jpg", System.Drawing.Imaging.ImageFormat.Jpeg);
```

We can select different file formats, and can also use things like a textbox or a full “save dialog” to allow the user to enter a file name.

Homework Assignment

As a homework assignment, you'll combine the two lessons from today to create an application that draws concentric circles with random colors each time the user clicks a button.

Some resources:

http://msdn2.microsoft.com/en-us/library/ms173064.aspx

http://www.samspublishing.com/library/content.asp?b=STY_Csharp_24hours&seqNum=105&rl=1