

**Module 05**

# **User Interfaces**

**CS 106 Winter 2018**

# **UI is a big topic**

**GBDA 103: User Experience Design**

# **UI is a big topic**

**GBDA 103: User Experience Design**

**CS 349: User Interfaces**

**CS 449: Human-Computer Interaction**

**MSCI 343: Human-Computer Interaction**

**DAC 309: User Experience Design**



San Jose, CA, USA

May 7-12

- Home
- Attending
- Planning
- Authors
- Organizers
- Sponsoring
- Recruiting
- Exhibiting
- Streaming

## CHI4GOOD



## HOME

### Welcome

Welcome to ACM CHI 2016: the top conference for Human-Computer Interaction. CHI will take place from May 7 – 12 at San Jose, CA, USA.

CHI 2016 Technical Program Preview




## QUICK LINKS

[Conference Registration](#)

[Conference Web App](#)

[CHI 2016 iOS and Android Apps](#)

# MONDAY 14:30 PM - 15:50 PM

## 220A Panel: User Experience (UX) in India

User Experience (UX) in India - 'We are Not Like This Only' - We are World Class and Much More!

*Apala Lahiri Chavan, Girish Prabhu, Sarit Arora, Janaki Kumar, Sudhindra V*

## 220B alt.chi: Critical Theory and Pedagogy

*Chair: Silvia Lindtner*

The User Experience in Zen and the Art of Motorcycle Maintenance

*Simon Harper*

Meaning Reconstruction as an Approach to Analyze Critical Dimensions of HCI Research

*Colin M Gray, Austin L Toombs, Christian McKay*

Critical Realist HCI

*Christopher Frauenberger*

Making the Case for an Existential Perspective in HCI Research on Mortality and Death

*Victor Kaptelinin*

## 210D Course: C01

Research Methods for Child Computer Interaction (2/2)

*Janet C Read, Shuli Gilutz*

## 210H Course: C03

Designing with the Mind in Mind: The Psychological Basis for UI Design Guidelines (2/2)

*Jeff A Johnson*

## 210C Course: C05

Introduction To Human Computer Interaction (2/2)

*Jonathan Lazar, Simone D J Barbosa*

## 210G Course: C08

Faceless Interaction - A Conceptual Examination of the Notion of Interface: Past, Present, and Future

*Lars-Erik Janlert, Erik Stolterman*

Five Provocations for Ethical HCI Research

*Barry Brown, Alexandra Weilenmann, Donald McMillan, Airi Lampinen*

Acting with Technology: Rehearsing for Mixed-Media Live Performances

*Louise Barkhuus, Chiara Rossitto*

## 112 SIG: Refugees and HCI

Refugees and HCI SIG: The Role of HCI in Responding To the Refugee Crisis

*Reem Talhouk, Syed Ishtiaque Ahmed, Volker Wulf, Clara Crivellara,*

*Vasilis Vlachokyriakos, Patrick Olivier*

## 114 Case Studies: Tools for Workers

*Chair: Pernille Bjorn*

Untethered Workspaces: A Zones Concept Towards Supporting Operator Movements in Control Rooms

*Veronika Domova, Saad Azhar, Maria Ralph, Jonas Brönmark*

From Two CSCW Frameworks to User Requirements Definition for a Retail Planning Collaborative Software

*Grégory Petit, Justin Sales*

Interactive Colormapping: Enabling Multiple Data Range and Detailed Views of Ocean Salinity

*Francesca Samsel, Sebastian Klaassen, Mark Petersen, Terece L Turton,*

*Greg D Abram, David H Rogers, James Ahrens*

Designing the Alarm Management User Experience for Patient Monitoring

*Sharoda A Paul, Alexander K Carroll, Stephen Treacy*

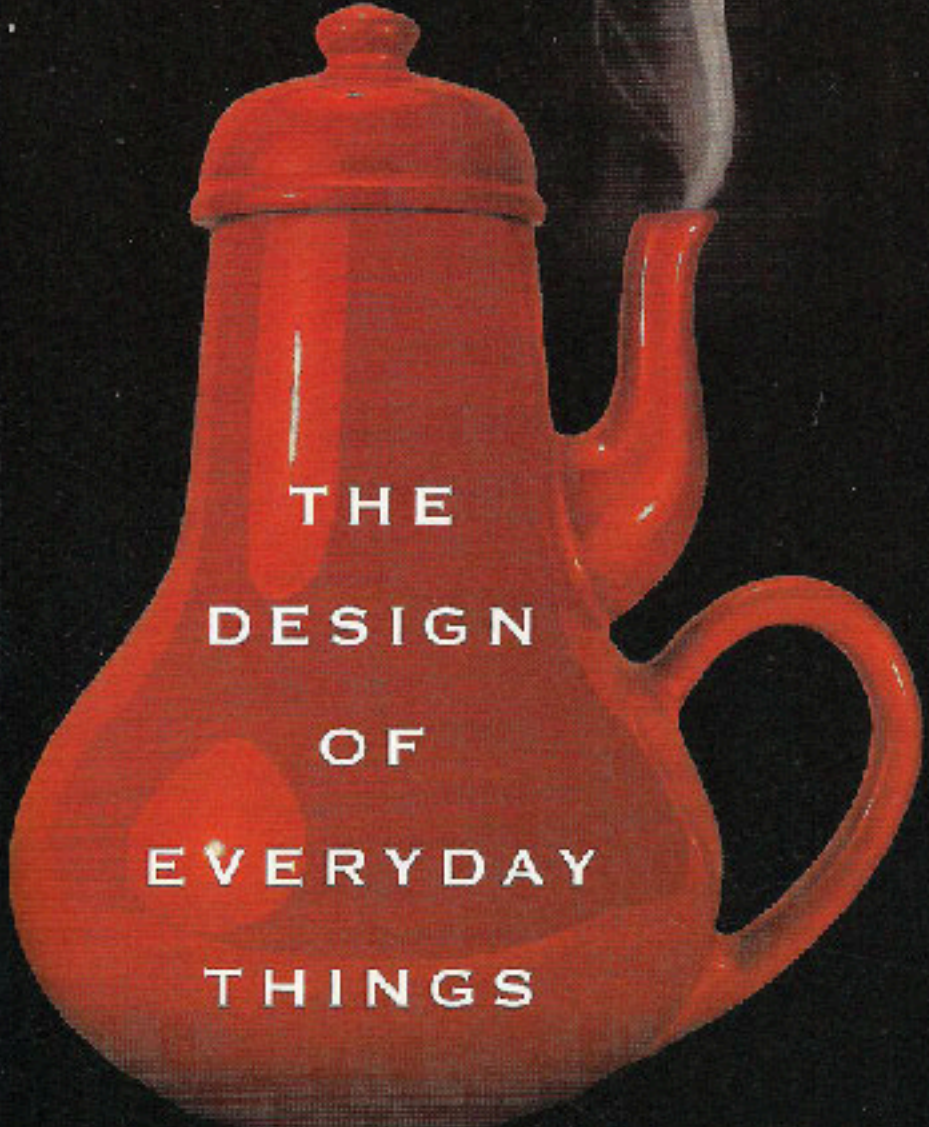
## LL21C Papers: Computer Supported Parenting



**MORE THAN 100,000 COPIES SOLD**  
**WITH A NEW INTRODUCTION BY THE AUTHOR**

"Design may be  
our top competitive  
edge. This book is a  
joy—fun and of the  
utmost importance."

**TOM PETERS**



**THE  
DESIGN  
OF  
EVERYDAY  
THINGS**

Previously published as *THE PSYCHOLOGY OF EVERYDAY THINGS*

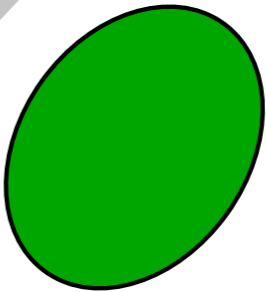
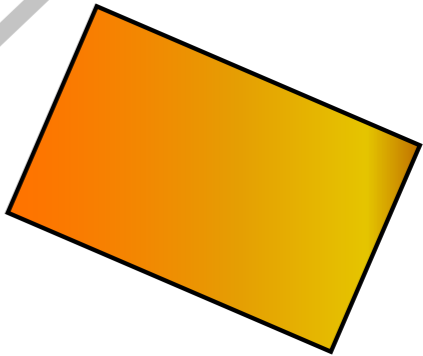
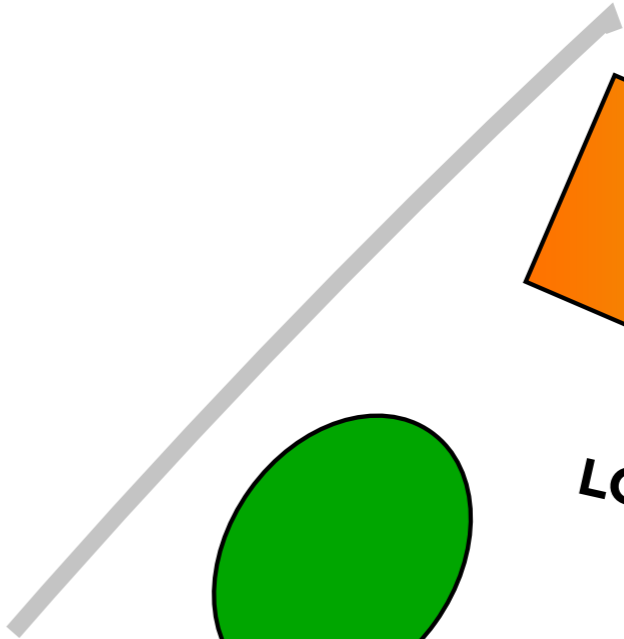
**DONALD A. NORMAN**

**AUTHOR OF EMOTIONAL DESIGN**

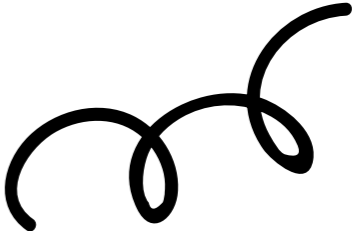
**How do programmers think about user interfaces?**

**What tools and techniques do they use to create user interfaces?**

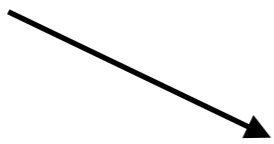




LOREM IPSUM



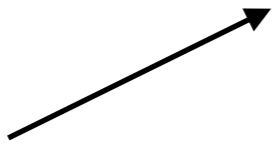
mouseX



mouseY



key



# Topics

**Model-view-controller paradigm**

**Direct manipulation**

**User interface toolkits**

**Building interfaces with ControlP5**

# Model-View-Controller (MVC)

A standard *paradigm* for describing the components of an interactive program.

**Model:** the underlying object or data being manipulated by the program.

**View:** the means by which the model is communicated to the user.

**Controller:** the means by which the user is able to manipulate the model.

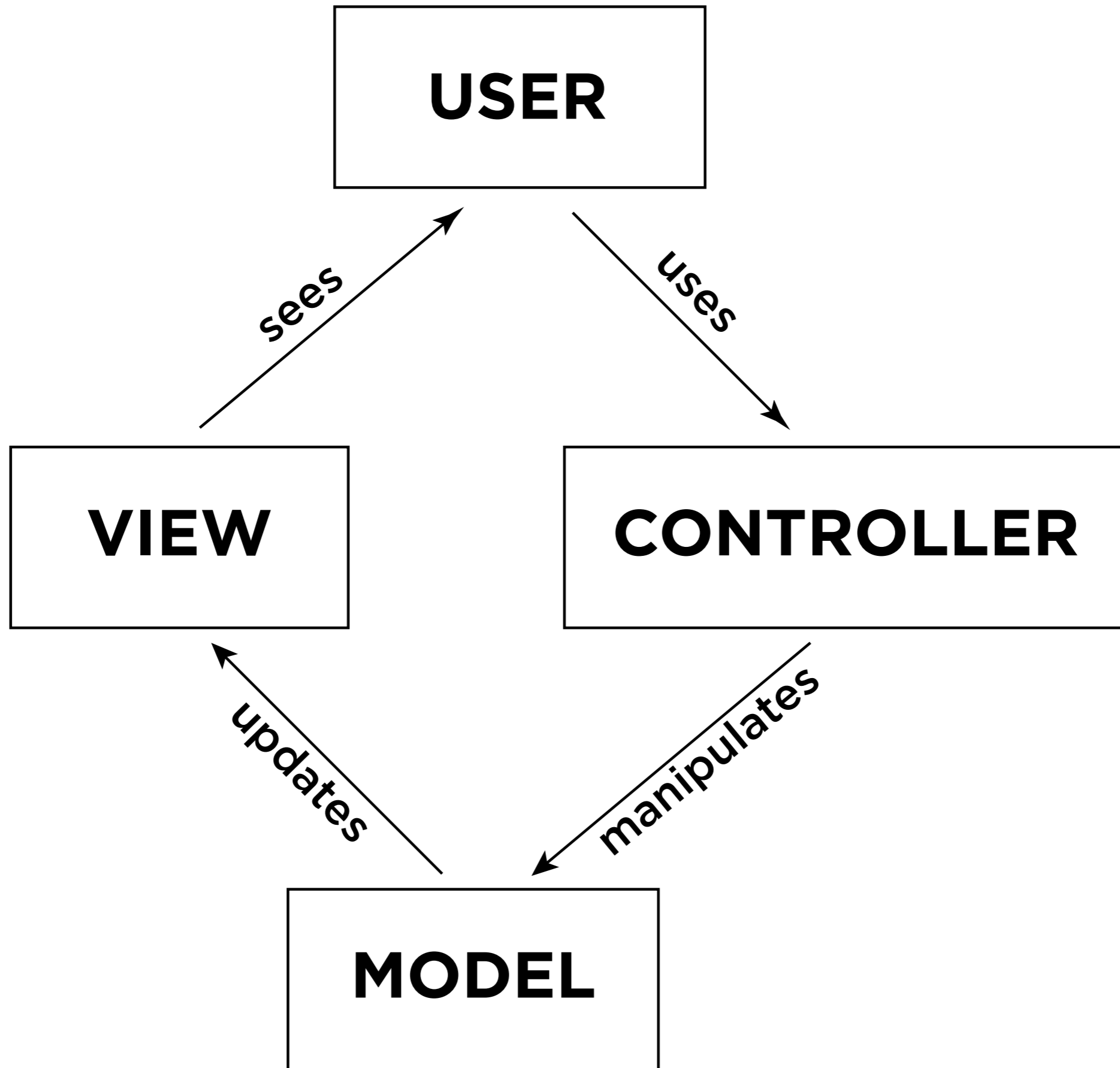
**VIEW**

**CONTROLLER**

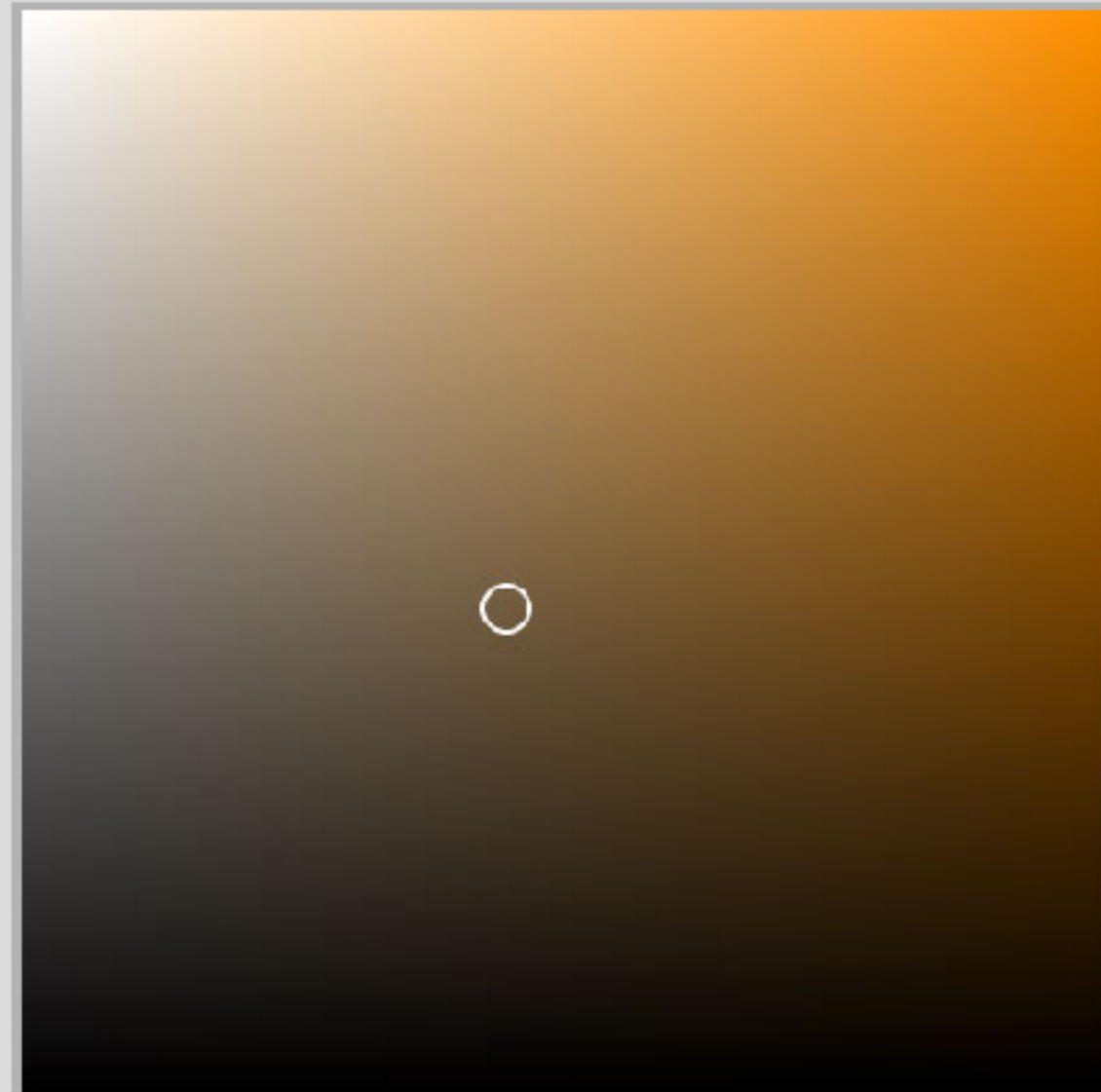
**MODEL**

*updates*

*manipulates*



# Color Picker (Foreground Color)



new



current

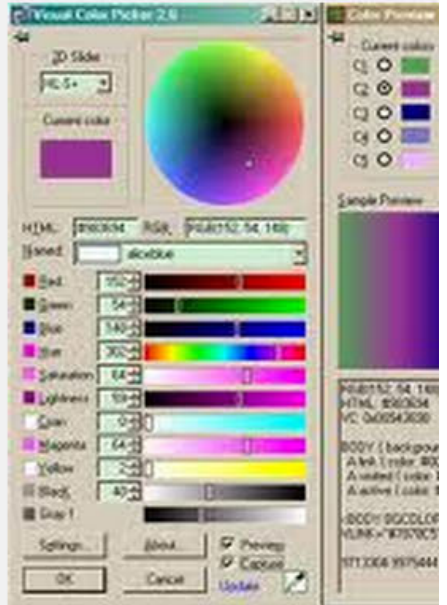
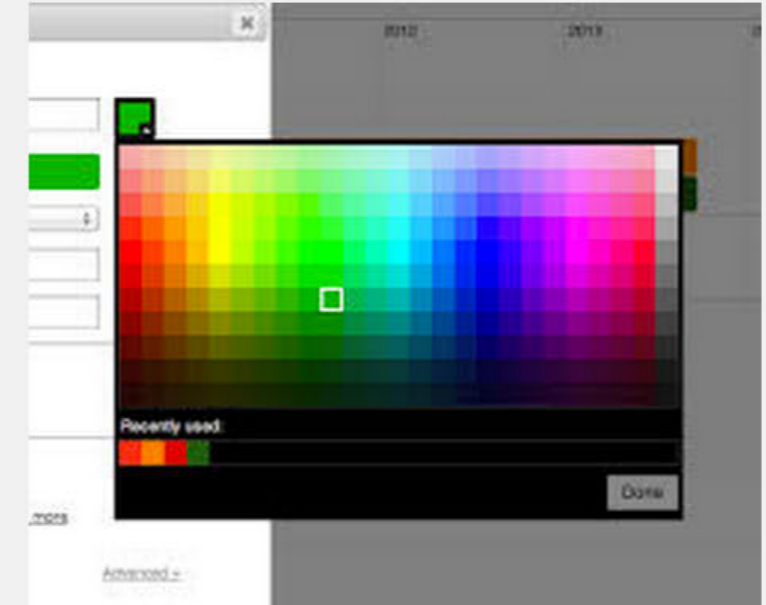
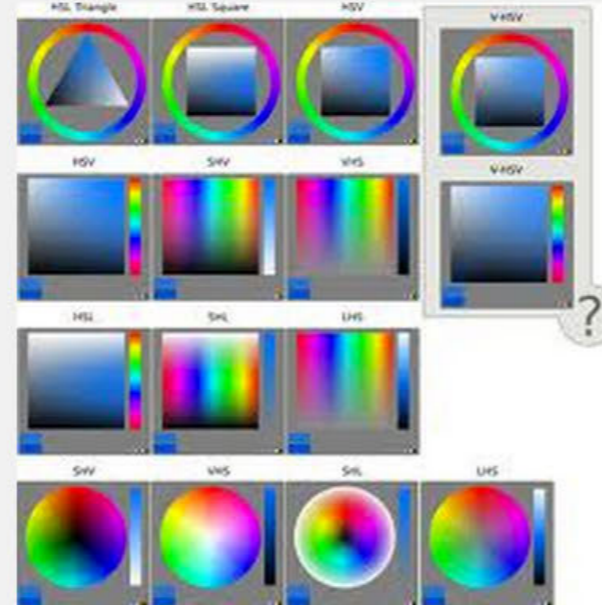
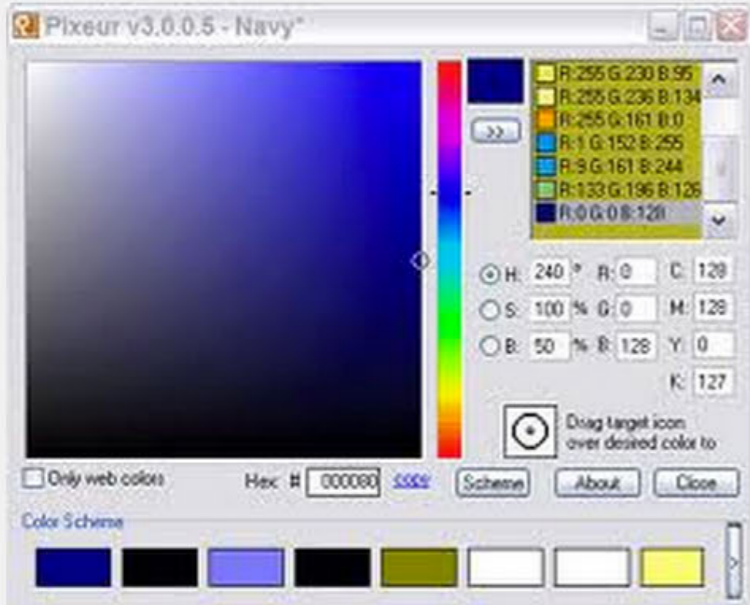
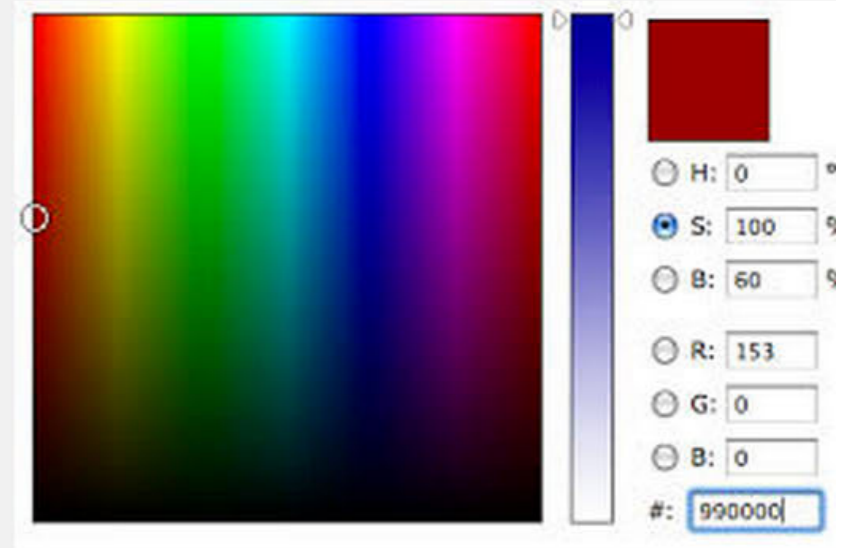
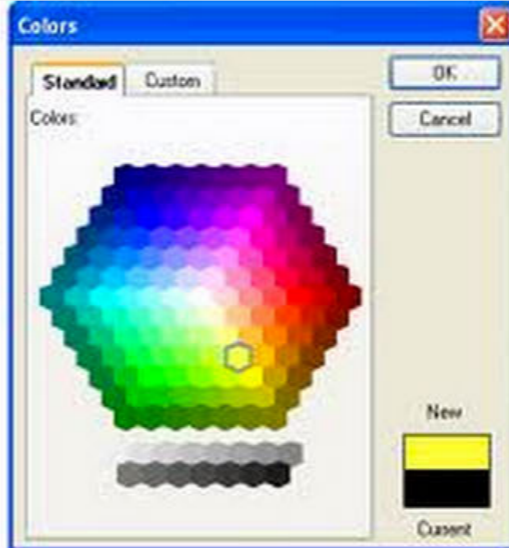
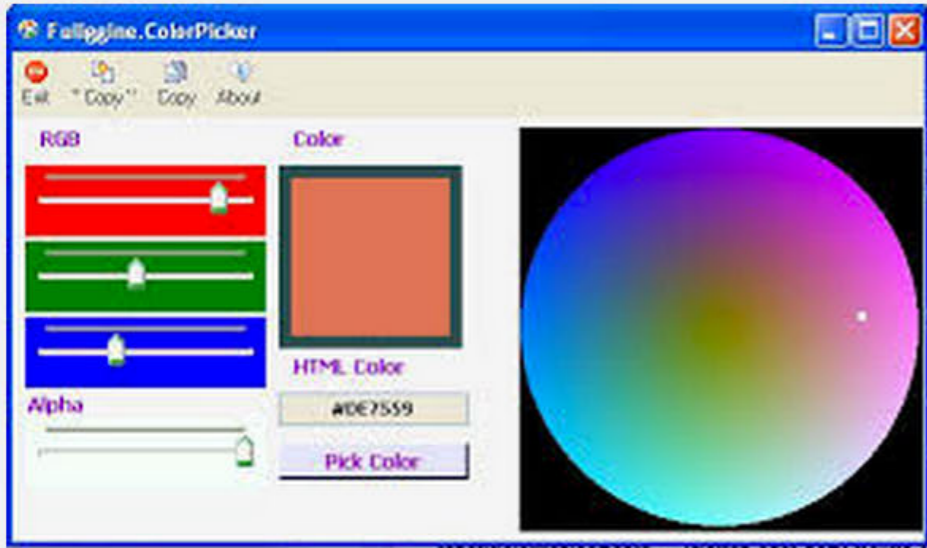
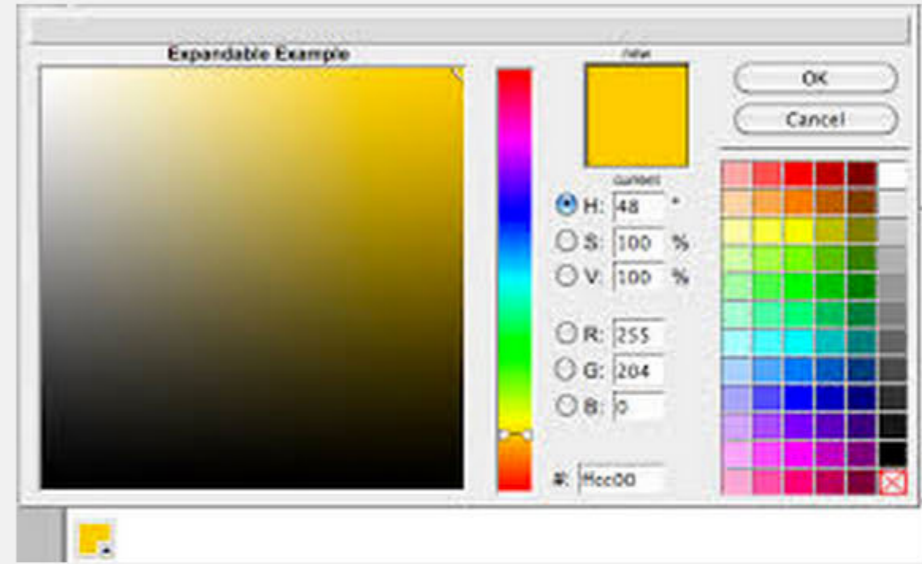


- OK
- Cancel
- Add to Swatches
- Color Libraries

Only Web Colors

<input checked="" type="radio"/> H: 35 °	<input type="radio"/> L: 41
<input type="radio"/> S: 45 %	<input type="radio"/> a: 6
<input type="radio"/> B: 45 %	<input type="radio"/> b: 21
<input type="radio"/> R: 115	C: 46 %
<input type="radio"/> G: 93	M: 54 %
<input type="radio"/> B: 63	Y: 77 %
# 735d3f	K: 29 %

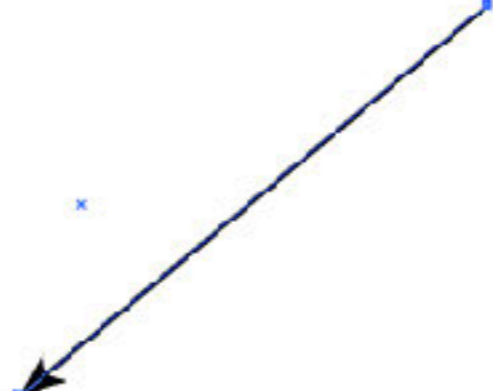
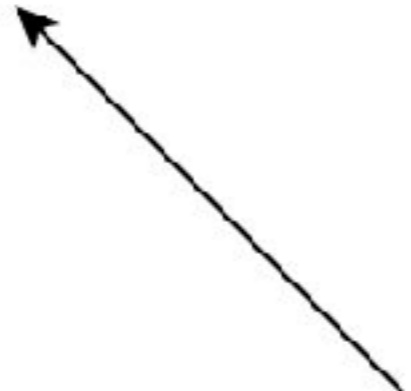




**VIEW**

**CONTROLL**

**MODEL**



Color Color Guide

R

G

B

#

Transform Align Pathfinder

Align Objects:

Distribute Objects:

Distribute Spacing: 0 pt

Align To:

Character Paragraph OpenType

Touch Type Tool

Gotham

Bold

24 pt (28.8 pt)

Auto VA 0

100% T 100%

0 pt T 0°

English: USA

Color Color Guide

R

G

B

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Generator: Adobe Illustrator 18.1.1, SVG Export Plug-In . SVG Version: 6.00 Build 0) -->
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN" "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg version="1.0" id="Layer_1" xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" x="0px" y="0px"
width="960px" height="560px" viewBox="0 0 960 560" style="enable-background:new 0 0 960 560;" xml:space="preserve">
<rect x="248.6" y="96.9" style="fill:#FFFFFF;" width="435.5" height="254"/>
<rect x="362.5" y="271.2" style="fill:#FFFFFF;stroke:#000000;stroke-miterlimit:10;" width="137.1" height="66.9"/>
<text transform="matrix(1 0 0 1 385.5725 311.3098)" style="font-family:'Gotham-Bold'; font-size:24;">MODEL</text>
<rect x="259.9" y="107.5" style="fill:#FFFFFF;stroke:#000000;stroke-miterlimit:10;" width="125" height="66.9"/>
<text transform="matrix(1 0 0 1 287.795 147.6002)" style="font-family:'Gotham-Bold'; font-size:24;">VIEW</text>
<rect x="464.8" y="107.5" style="fill:#FFFFFF;stroke:#000000;stroke-miterlimit:10;" width="208.3" height="66.9"/>
<text transform="matrix(1 0 0 1 482.1842 147.6002)" style="font-family:'Gotham-Bold'; font-size:24;">CONTROLLER</text>
<g>
  <g>
    <line style="fill:none;stroke:#000000;stroke-miterlimit:10;" x1="339.1" y1="185" x2="419.5" y2="265.5"/>
    <g>
      <polygon points="334.1,180 338.2,189.9 339.4,185.3 344,184.1" />
    </g>
  </g>
</g>
<g>
  <g>
    <line style="fill:none;stroke:#000000;stroke-miterlimit:10;" x1="458" y1="261.7" x2="555.8" y2="180"/>
    <g>
      <polygon points="452.6,266.3 462.9,263 458.4,261.4 457.6,256.8" />
    </g>
  </g>
</g>
</svg>
```

Distribute Objects:

Distribute Spacing: 0 pt

Align To:

100%

0 pt

0°

English: USA

```
color the_colour;
```

```
void setup()  
{  
  size( 200, 200 );  
}
```

```
void draw()  
{  
  background( the_colour );  
}
```

```
void mouseMoved()  
{  
  float r = map( mouseX, 0, width, 0, 255 );  
  float g = map( mouseY, 0, height, 0, 255 );  
  the_colour = color( r, g, 0 );  
}
```

```
color the_colour;
```

## Model

```
void setup()
```

```
{
```

```
  size( 200, 200 );
```

```
}
```

```
void draw()
```

```
{
```

```
  background( the_colour );
```

```
}
```

```
void mouseMoved()
```

```
{
```

```
  float r = map( mouseX, 0, width, 0, 255 );
```

```
  float g = map( mouseY, 0, height, 0, 255 );
```

```
  the_colour = color( r, g, 0 );
```

```
}
```

```
color the_colour;
```

**Model**

```
void setup()  
{  
  size( 200, 200 );  
}
```

**View**

```
void draw()  
{  
  background( the_colour );  
}
```

```
void mouseMoved()  
{  
  float r = map( mouseX, 0, width, 0, 255 );  
  float g = map( mouseY, 0, height, 0, 255 );  
  the_colour = color( r, g, 0 );  
}
```

```
color the_colour;
```

**Model**

```
void setup()  
{  
  size( 200, 200 );  
}
```

**View**

```
void draw()  
{  
  background( the_colour );  
}
```

**Controller**

```
void mouseMoved()  
{  
  float r = map( mouseX, 0, width, 0, 255 );  
  float g = map( mouseY, 0, height, 0, 255 );  
  the_colour = color( r, g, 0 );  
}
```

# Direct Manipulation

The controller is coupled to the view (or equal to the view)

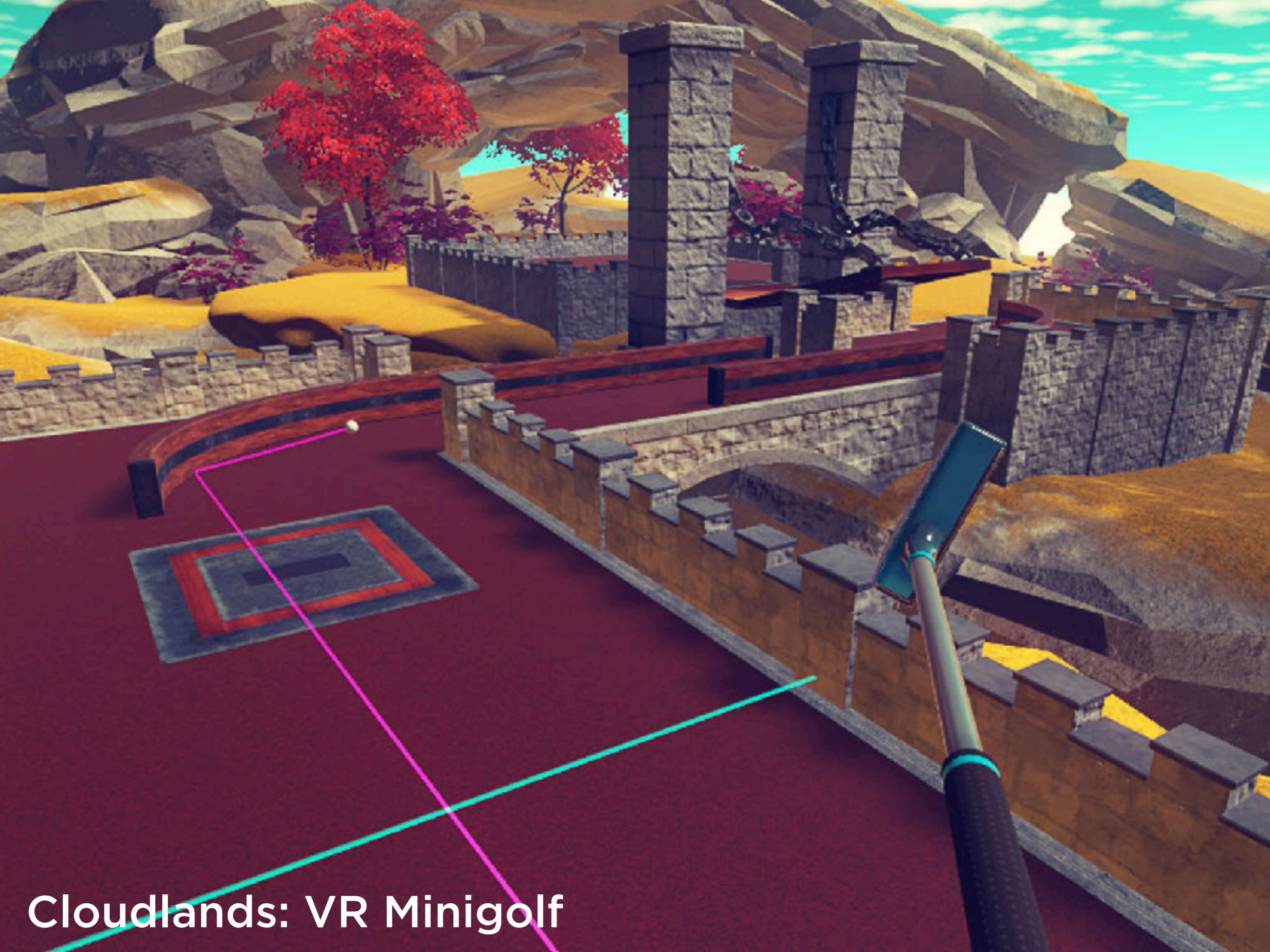
Interaction is continuous and incremental.



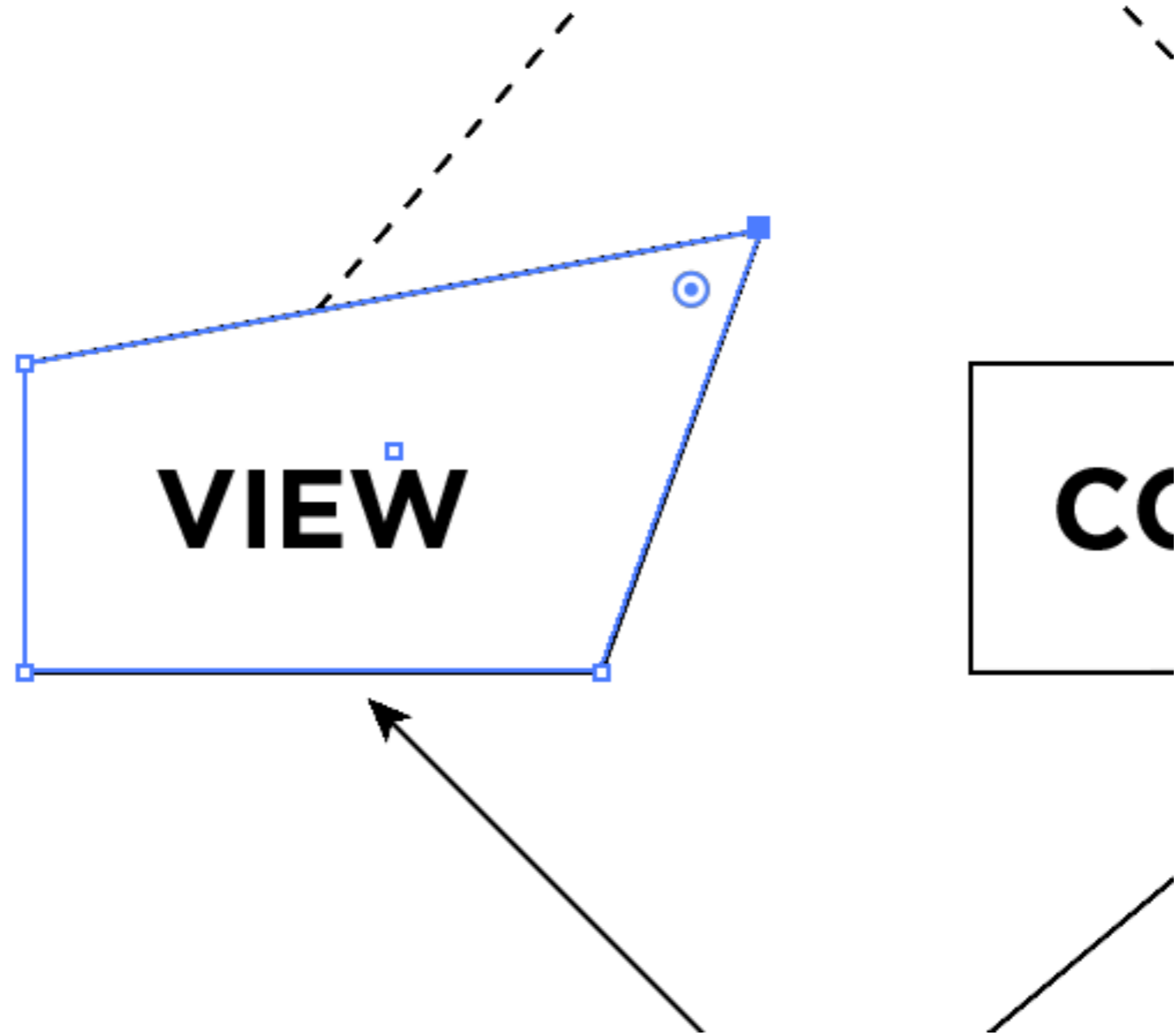


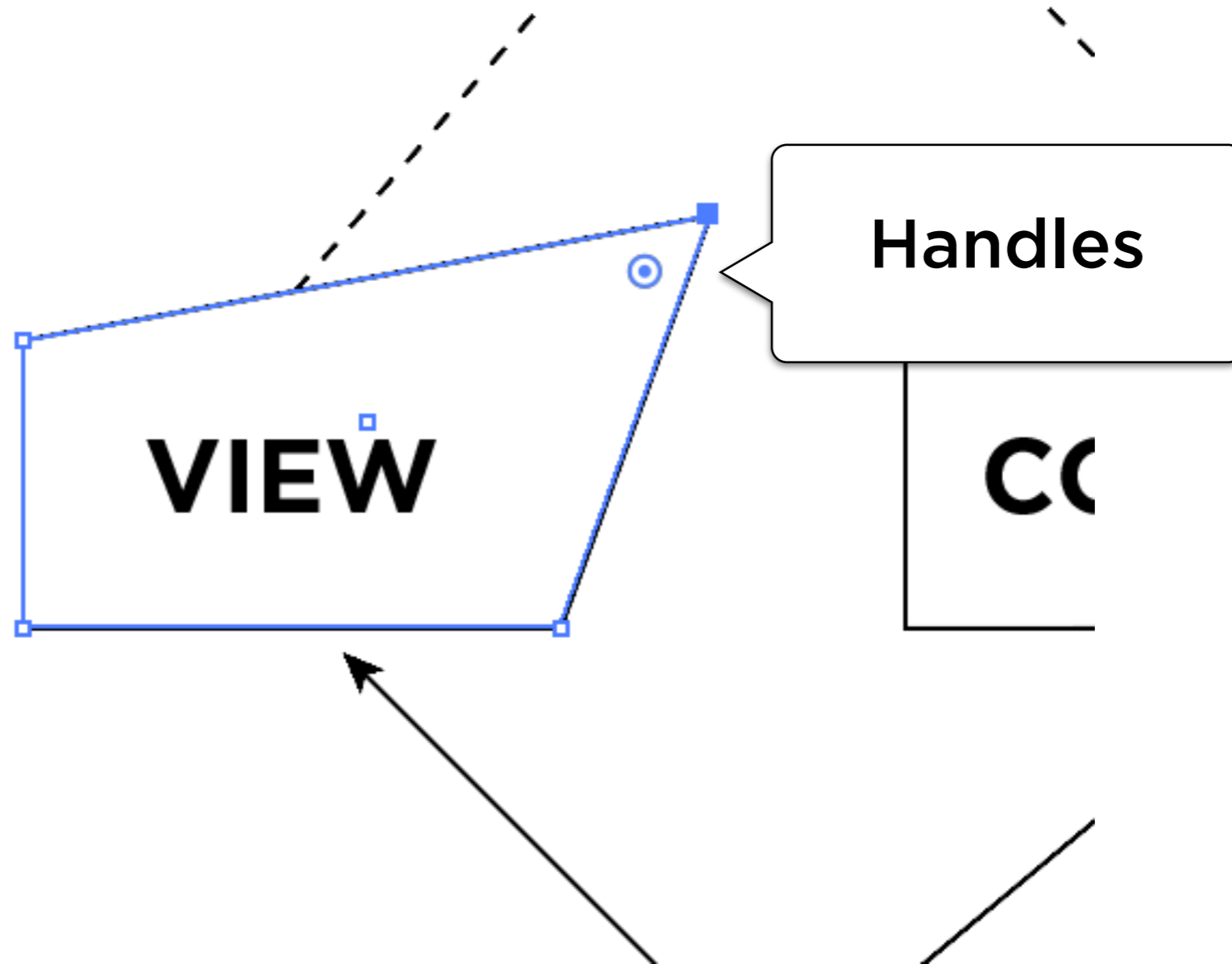
Craig S. Kaplan - Computer (White to Move) — Edited





Cloudlands: VR Minigolf





# Hit Testing

Every on-screen element that can be manipulated needs a *hit test*—a way to determine if the mouse is over that element.

```
ellipse( cx, cy, 2 * rad, 2 * rad );
```

**Draw a circle with centre (cx, cy) and radius rad.**

```
ellipse( cx, cy, 2 * rad, 2 * rad );  
  
if ( dist( mouseX, mouseY, cx, cy ) <= rad ) {  
    ...  
}
```

**Hit test for the same circle.**

```
ellipse( cx, cy, 2 * rad, 2 * rad );
```

```
if ( dist( mouseX, mouseY, cx, cy ) <= rad ) {  
    ...  
}
```

```
rect( ax, ay, w, w );
```

```
if ( (mouseX >= ax) && (mouseX <= (ax+w))  
    && (mouseY >= ay) && (mouseY <= (ay+w)) ) {  
    ...  
}
```



# Direct Manipulation and classes

```
class InteractiveThingy
{
    // Fields (i.e., part of the Model)

    void drawSelf()
    {
        // Draw this object in the sketch (View)
    }

    boolean hitTest( float x, float y )
    {
        // Is point (x,y) inside this object? (Controller)
    }
}
```

# Handling events

```
boolean active;
```

```
void mousePressed()  
{  
    float d = dist( cx, cy, mouseX, mouseY );  
    if ( d < rad ) {  
        active = true;  
    }  
}
```

```
void mouseDragged()  
{  
    if ( active ) {  
        cx = mouseX;  
        cy = mouseY;  
    }  
}
```

```
void mouseReleased()  
{  
    active = false;  
}
```

# Handling events

`boolean active;`

**Are we dragging the circle?**

```
void mousePressed()
{
    float d = dist( cx, cy, mouseX, mouseY );
    if ( d < rad ) {
        active = true;
    }
}
```

```
void mouseDragged()
{
    if ( active ) {
        cx = mouseX;
        cy = mouseY;
    }
}
```

```
void mouseReleased()
{
    active = false;
}
```

# Handling events

`boolean active;`

**Are we dragging the circle?**

```
void mousePressed()
```

```
{
```

```
float d = dist( cx, cy, mouseX, mouseY );
```

```
if ( d < rad ) {
```

```
    active = true;
```

```
}
```

```
}
```

**Hit test**

```
void mouseDragged()
```

```
{
```

```
    if ( active ) {
```

```
        cx = mouseX;
```

```
        cy = mouseY;
```

```
    }
```

```
}
```

```
void mouseReleased()
```

```
{
```

```
    active = false;
```

```
}
```

# Handling events

`boolean active;`

**Are we dragging the circle?**

```
void mousePressed()
```

```
{
```

```
float d = dist( cx, cy, mouseX, mouseY );
```

```
if ( d < rad ) {
```

```
    active = true;
```

```
}
```

```
}
```

**Hit test**

```
void mouseDragged()
```

```
{
```

```
if ( active ) {
```

```
    cx = mouseX;
```

```
    cy = mouseY;
```

```
}
```

```
}
```

**Controller updates the model**

```
void mouseReleased()
```

```
{
```

```
    active = false;
```

```
}
```

# Handling events

```
boolean active;
```

```
void mousePressed()  
{  
    float d = dist( cx, cy, mouseX, mouseY );  
    if ( d < rad ) {  
        active = true;  
    }  
}
```

```
void mouseDragged()  
{  
    if ( active ) {  
        cx += mouseX - pmouseX;  
        cy += mouseY - pmouseY;  
    }  
}
```

```
void mouseReleased()  
{  
    active = false;  
}
```

If we have an interface with multiple elements, we need a way to keep track of which one was hit.

```
boolean circle_active = false;  
boolean square_active = false;
```

```
void draw()  
{  
    drawCircle( ... );  
    drawSquare( ... );  
}
```

```
void mousePressed()  
{  
    circle_active = false;  
    square_active = false;  
  
    if( hitTestCircle( ... ) ) {  
        circle_active = true;  
    } else if( hitTestSquare( ... ) ) {  
        square_active = true;  
    }  
}
```

If we have an interface with multiple elements, we need a way to keep track of which one was hit.

```
boolean circle_active = false;  
boolean square_active = false;
```

```
void draw()  
{  
    drawSquare( ... );  
    drawCircle( ... );  
}
```

```
void mousePressed()  
{  
    circle_active = false;  
    square_active = false;  
  
    if( hitTestCircle( ... ) ) {  
        circle_active = true;  
    } else if( hitTestSquare( ... ) ) {  
        square_active = true;  
    }  
}
```



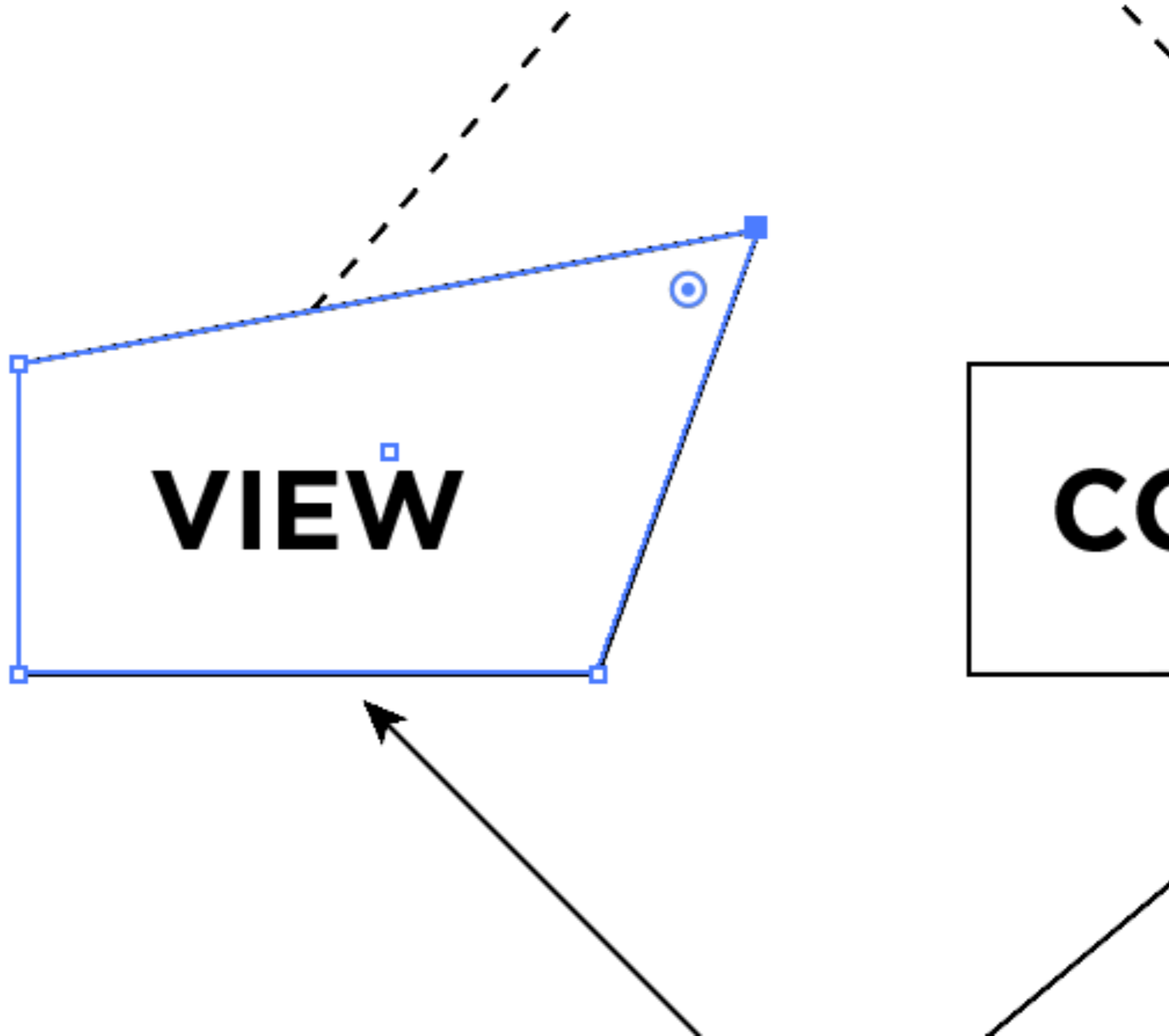
**If we have an interface with multiple elements, we need a way to keep track of which one was hit.**

```
Circle[] some_circles;  
int active = -1;
```

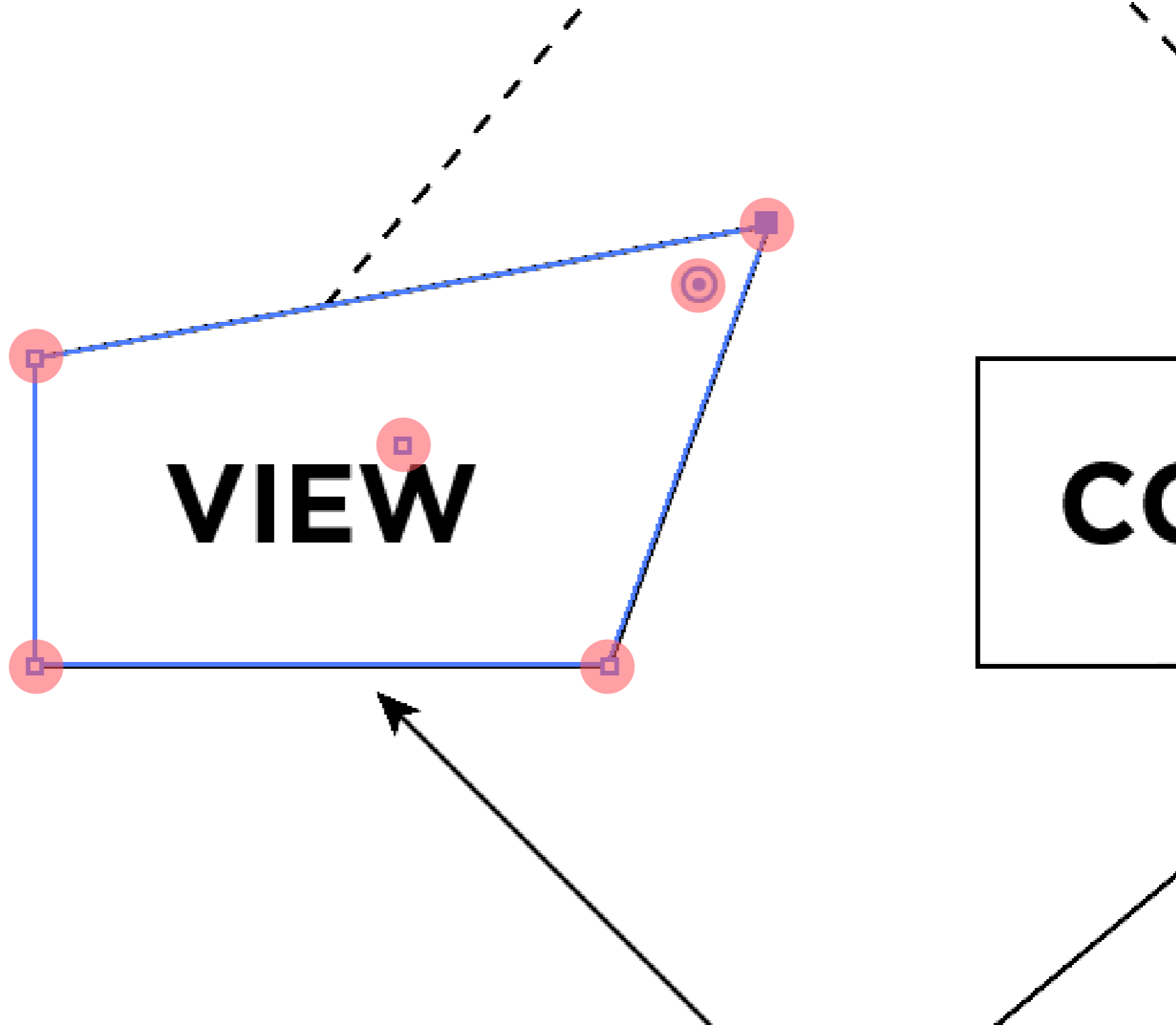
```
void draw()  
{  
    for( int idx = 0; idx < some_circles.length; ++idx ) {  
        drawCircle( some_circles[idx] );  
    }  
}
```

```
void mousePressed()  
{  
    active = -1;  
    for( int idx = some_circles.length - 1; idx >= 0; --idx ) {  
        if( hitTest( some_circles[idx] ) ) {  
            active = idx;  
            return;  
        }  
    }  
}
```

# Small handles



# Small handles

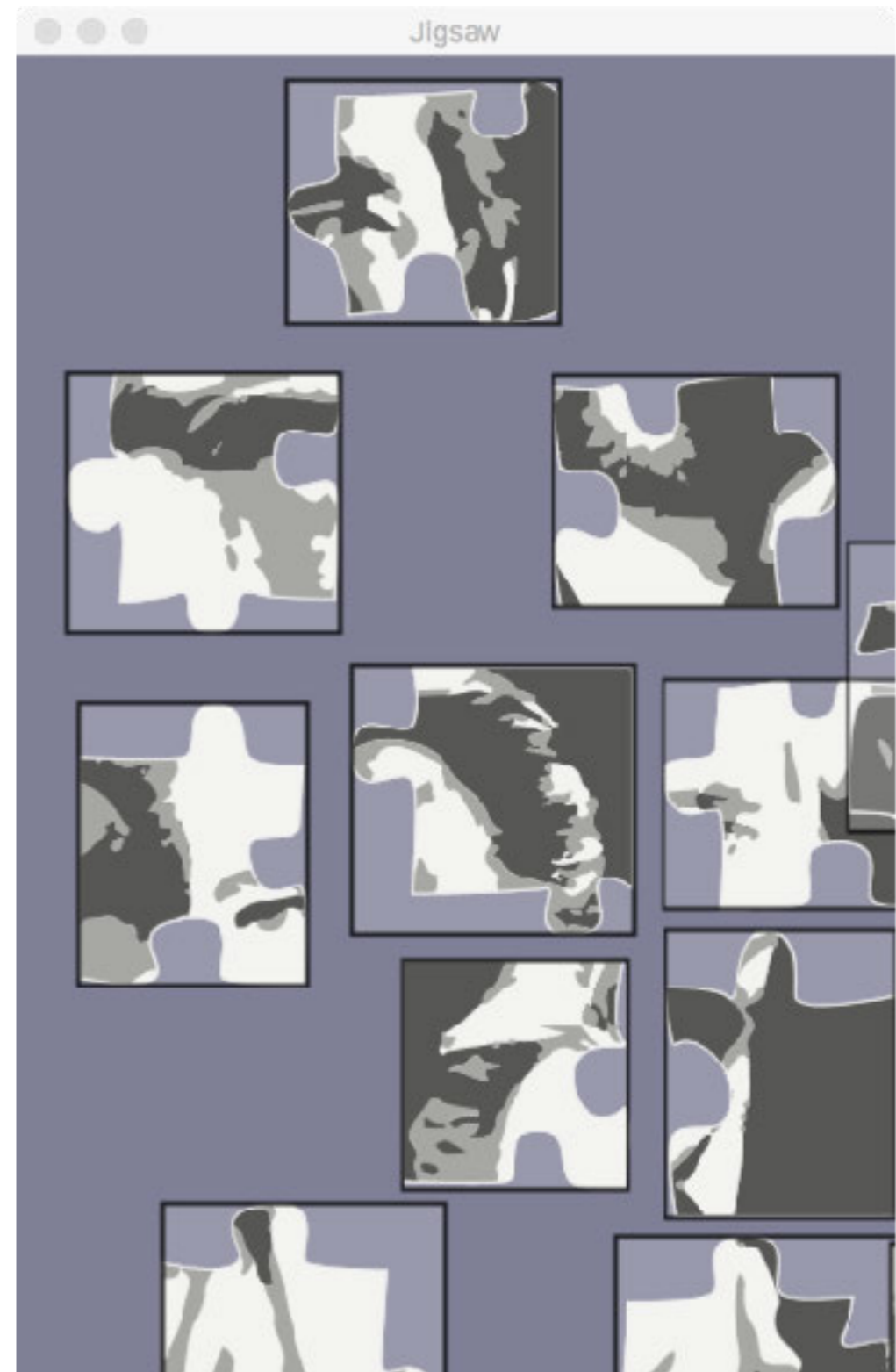
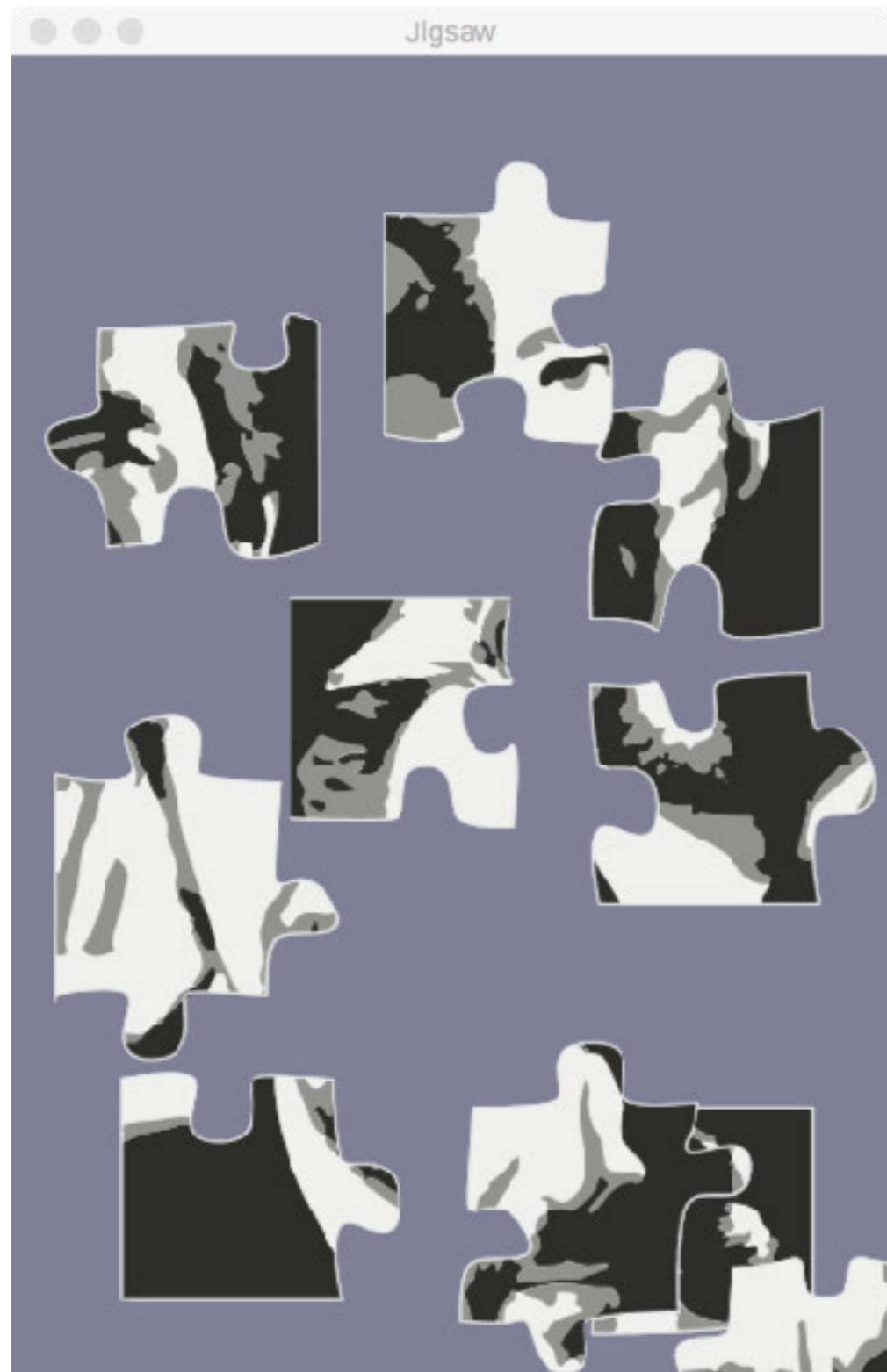


# Complex shapes

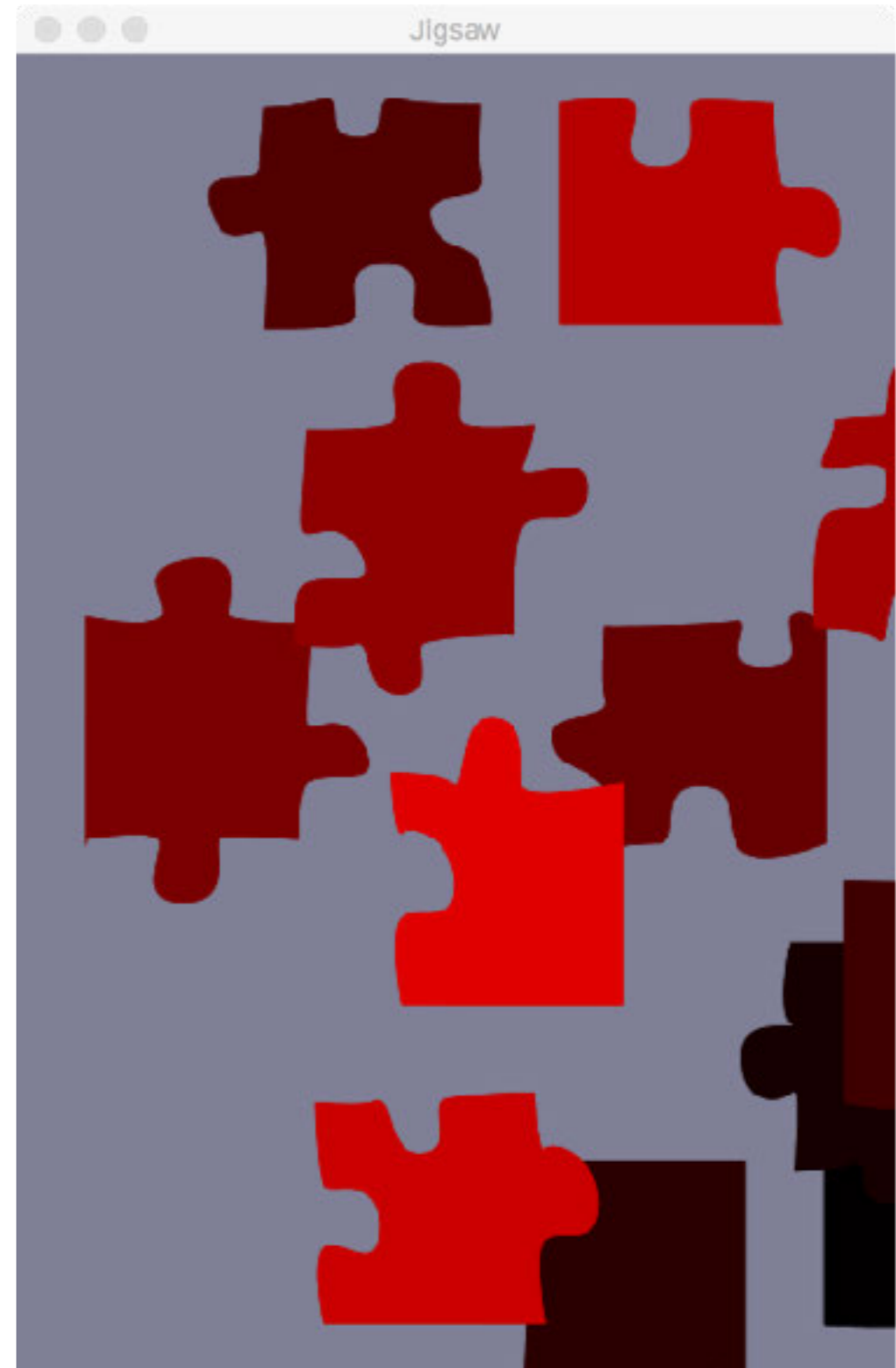


How can we hit test a shape with a complicated boundary?

# Proxy geometry



# Selection buffer



# Direct manipulation notes

**Shift objects using  $\text{mouseX} - \text{pmouseX}$  and  $\text{mouseY} - \text{pmouseY}$ , don't "teleport" them.**

**Draw objects from back-to-front, hit test them from front-to-back.**

**Make hit test region usable, regardless of how it's drawn.**

# Toolkits

Some interactions are so canonical that it makes sense to invent standardized “widgets” to handle them.

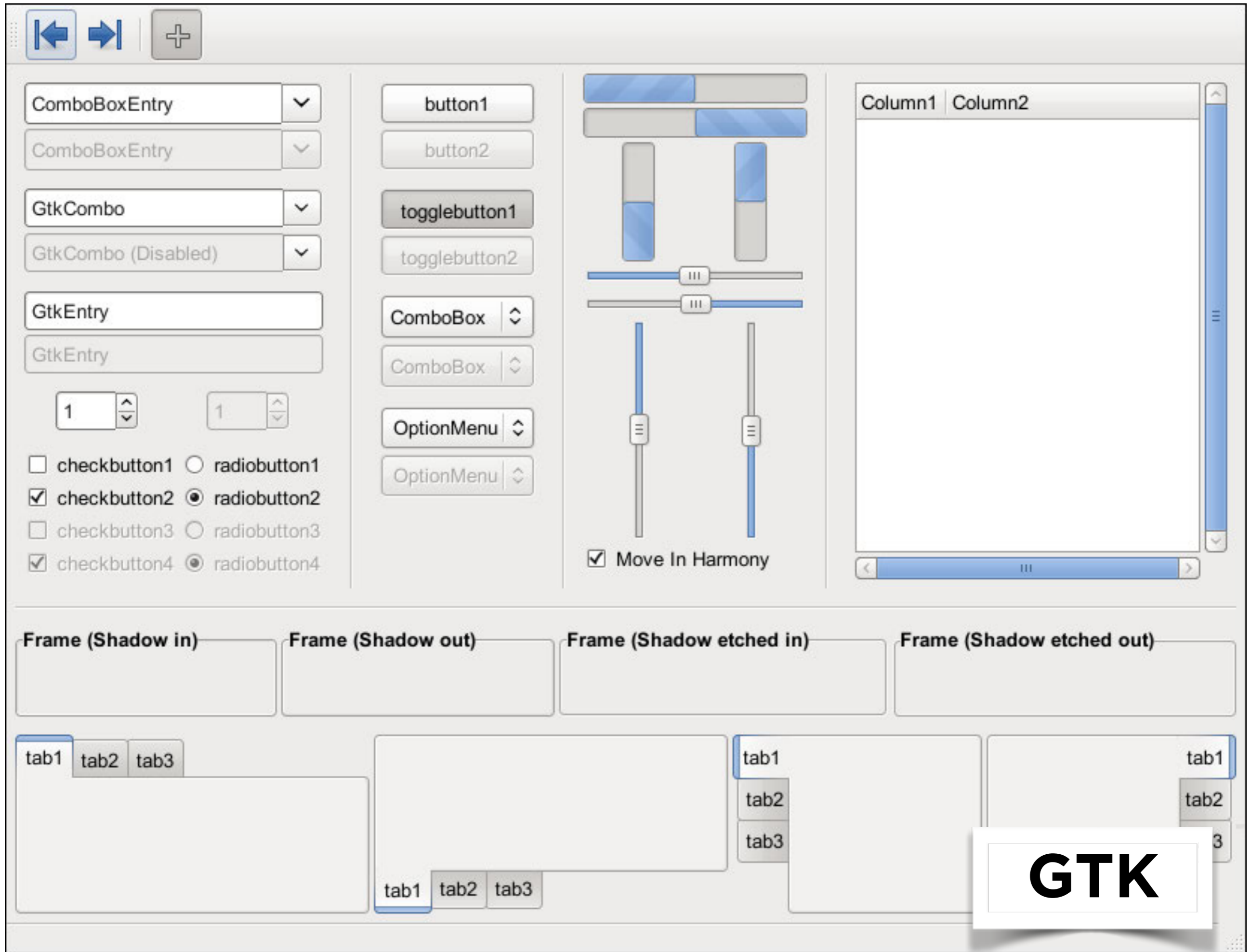
Perform an action: **Button**

Set a continuous value: **Slider**

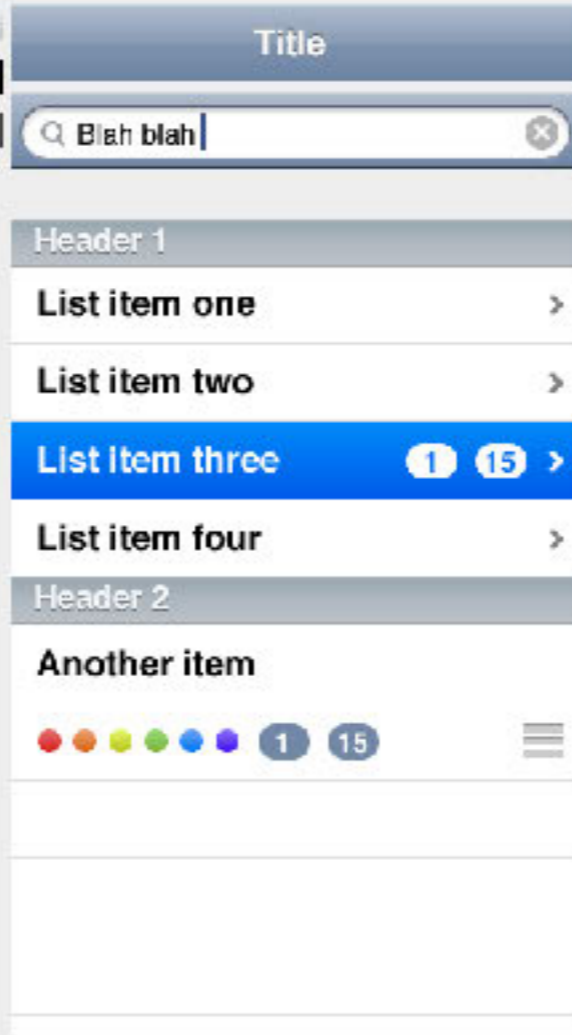
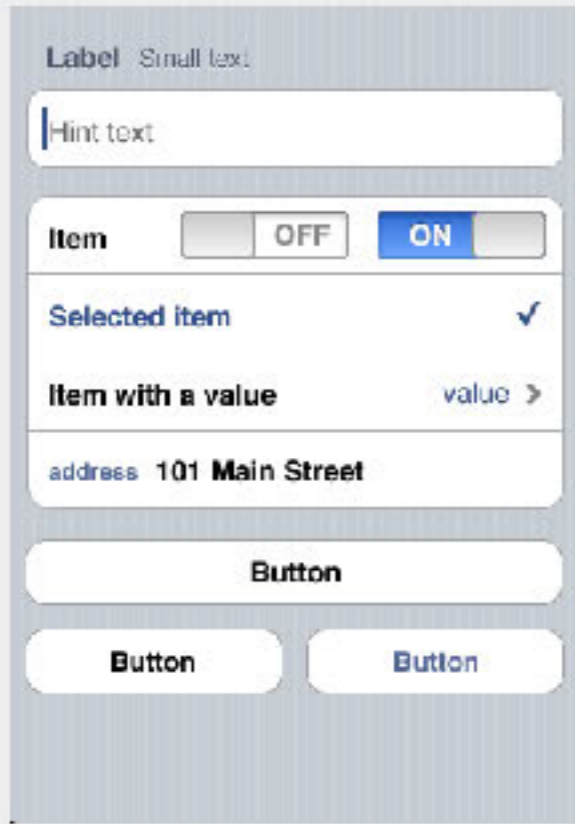
Enter text: **Text field**

**Classes and objects are perfect for this!**

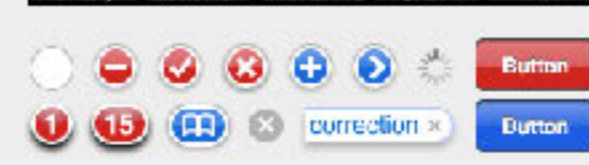
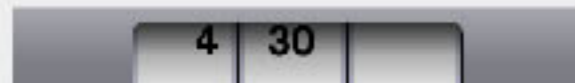
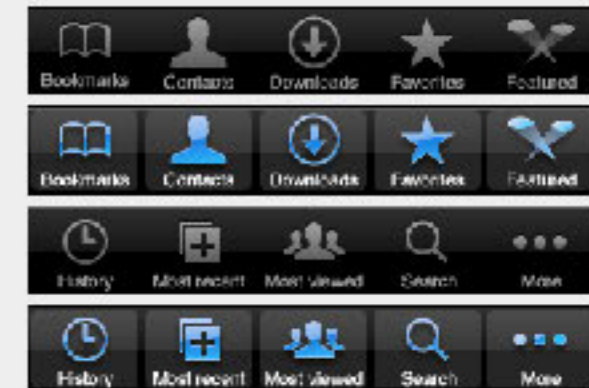
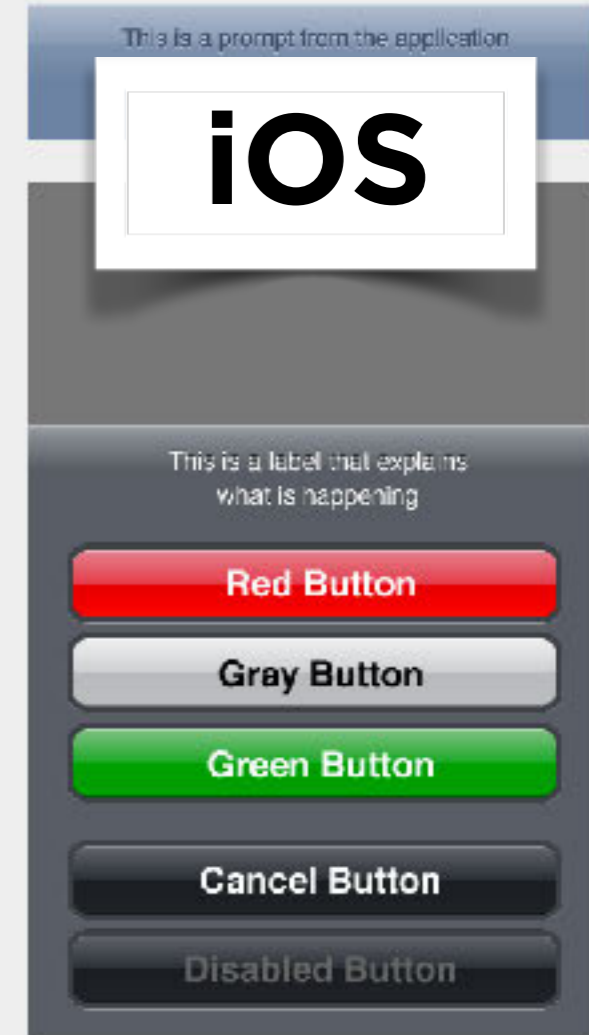
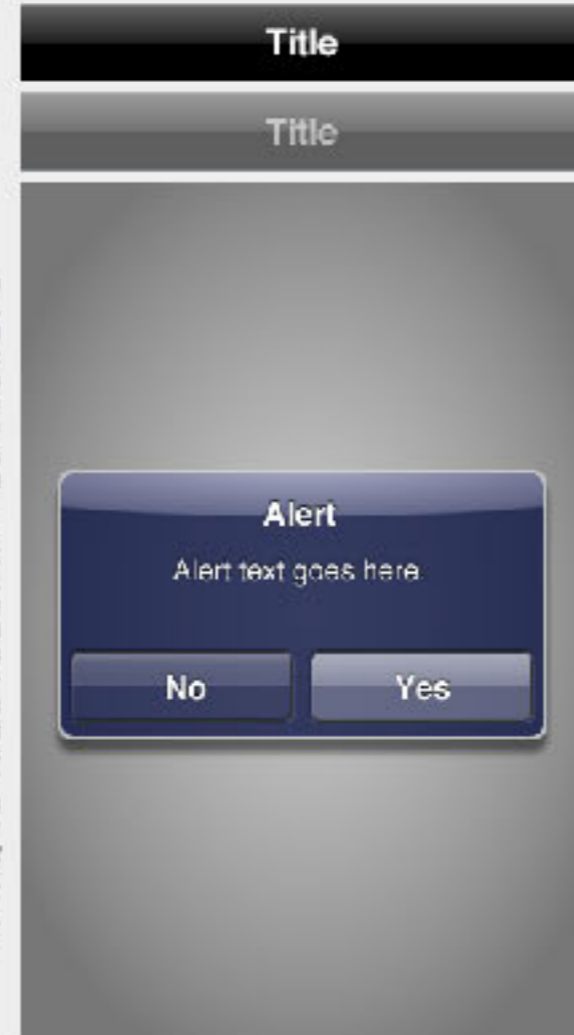




**GTK**



Q  
A  
B  
C  
D  
E  
F  
G  
H  
I  
J  
K  
L  
M  
N  
O  
P  
R  
S  
T  
U  
V  
W  
X  
Y  
Z  
F



# ControlP5

LISTBOX -

- ITEM 0
- ITEM 1
- ITEM 2
- ITEM 3
- ITEM 4
- ITEM 5
- ITEM 6
- ITEM 7

MULTILIST

- LEVEL1
- LEVEL2
  - LEVEL2 ITEM1
  - LEVEL2 ITEM2
  - LEVEL2 ITEM4
  - LEVEL2 ITEM5
  - LEVEL2 ITEM6

RADIOBUTTON



hello world

TEXTFIELD

BUTTON

BUTTON

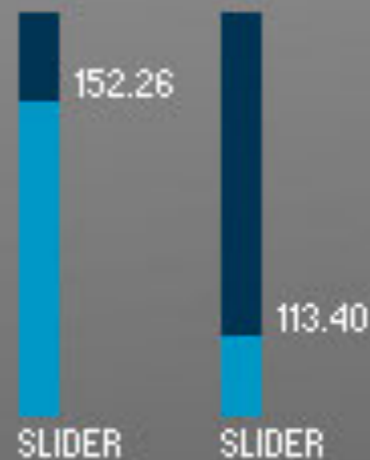
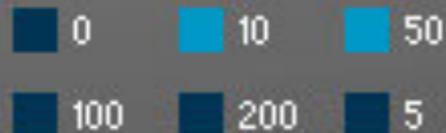
31

NUMBERBOX

232

NUMBERBOX

CHECKBOX



# Minimal ControlP5

```
import controlP5.*;

ControlP5 ui;

void setup()
{
  size( 500, 500 );

  ui = new ControlP5( this );
}
```

# Minimal ControlP5

```
import controlP5.*;
```

**Import directive**

```
ControlP5 ui;
```

```
void setup()
```

```
{
```

```
  size( 500, 500 );
```

```
  ui = new ControlP5( this );
```

```
}
```

# Minimal ControlP5

```
import controlP5.*;
```

**Import directive**

```
ControlP5 ui;
```

**Global “factory object”**

```
void setup()  
{  
  size( 500, 500 );  
  
  ui = new ControlP5( this );  
}
```

# Minimal ControlP5

```
import controlP5.*;
```

**Import directive**

```
ControlP5 ui;
```

**Global “factory object”**

```
void setup()
```

```
{
```

```
  size( 500, 500 );
```

```
  ui = new ControlP5( this );
```

```
}
```

**Initialize the library, “build the factory”**

# Minimal ControlP5

```
import controlP5.*;

ControlP5 ui;

void setup()
{
    size( 500, 500 );

    ui = new ControlP5( this );
    ui.addButton( "Hello!" );
}

void draw()
{ }
```



# Minimal ControlP5

```
import controlP5.*;
```

```
ControlP5 ui;
```

```
void setup()
```

```
{
```

```
  size( 500, 500 );
```

```
  ui = new ControlP5( this );
```

```
  ui.addButton( "Hello!" );
```

```
}
```

**Add a widget**

```
void draw()
```

```
{}
```

```
void setup()  
{  
    size( 500, 500 );  
  
    ui = new ControlP5( this );  
  
    Button hello = ui.addButton( "Hello!" );  
    hello.setPosition( 200, 200 );  
    hello.setSize( 120, 60 );  
}
```

```
void setup()
{
  size( 500, 500 );

  ui = new ControlP5( this );

  Button hello = ui.addButton( "Hello!" );
  hello.setPosition( 200, 200 );
  hello.setSize( 120, 60 );
}
```

**Hold on to an object that  
represents the button**

```
void setup()  
{  
  size( 500, 500 );  
  
  ui = new ControlP5( this );  
  
  Button hello = ui.addButton( "Hello!" );  
  hello.setPosition( 200, 200 );  
  hello.setSize( 120, 60 );  
}
```

**Set some of the button's properties**

```
void setup()
{
    size( 500, 500 );

    ui = new ControlP5( this );

    Button hello = ui.addButton( "Hello!" )
        .setPosition( 200, 200 );
        .setSize( 120, 60 );
}
```

```
class Point
{
    float x;
    float y;

    Point( float xIn, float yIn ) {
        x = xIn;
        y = yIn;
    }

    Point setX( float xIn ) {
        x = xIn;
        return this;
    }
}
```

```
void setup()
{
    size( 500, 500 );

    ui = new ControlP5( this );
    ui.setFont( createFont( "Gotham-Bold", 24 ) );

    Button hello = ui.addButton( "Hello!" )
        .setPosition( 200, 200 );
        .setSize( 120, 60 );
}
```

# Handling UI events

How do we discover when a button was pressed, and what can we do when that happens?

ControlP5 defines a new hook,  
`controlEvent()`.



```
void setup()
{
    size( 500, 500 );

    ui = new ControlP5( this );
    ui.setFont( createFont( "Gotham-Bold", 24 ) );

    Button hello = ui.addButton( "Hello!" )
        .setPosition( 200, 200 )
        .setSize( 120, 60 );
}

void controlEvent( ControlEvent ce )
{
    println( "Something happened!" );
}
```

```
void setup()
{
  size( 500, 500 );

  ui = new ControlP5( this );
  ui.setFont( createFont( "Gotham-Bold", 24 ) );

  Button hello = ui.addButton( "Hello!" )
    .setPosition( 200, 200 )
    .setSize( 120, 60 );
}
```

## **Name of the hook**

```
void controlEvent( ControlEvent ce )
{
  println( "Something happened!" );
}
```

```
void setup()
{
    size( 500, 500 );

    ui = new ControlP5( this );
    ui.setFont( createFont( "Gotham-Bold", 24 ) );

    Button hello = ui.addButton( "Hello!" )
        .setPosition( 200, 200 );
        .setSize( 120, 60 );
}
```

## Information about the event

```
void controlEvent( ControlEvent ce )
{
    println( "Something happened!" );
}
```

```
import controlP5.*;

ControlP5 ui;

Button b1;
Button b2;

void setup()
{
    size( 500, 500 );

    ui = new ControlP5( this );

    b1 = ui.addButton( "One" );
    b2 = ui.addButton( "Two" );
}
```

```
void draw()
{}

void controlEvent( ControlEvent ce )
{
    if( ce.isFrom( b1 ) ) {
        println( "One" );
    } else if( ce.isFrom( b2 ) ) {
        println( "Two" );
    }
}
```

## controlP5

A GUI (graphical user interface) library for processing.

## Download

Download controlP5 version 2.2.5 release 07/30/2015

[controlP5.zip](#)

This version has been tested with processing 2.2.1, for earlier version see the [download list](#).

About

Installation

Details

Features

Examples

JavaDoc Reference

Source Code

Back

## Older Versions

For older versions see the [download list](#) on the google code project page.

## FAQ

Frequently Asked Questions might have been answered in the processing forum. Have a look and search for controlP5 [here](#). Or file an [issue on github](#)

## Some projects using controlP5

[decode](#)  
[cop15 identity](#)  
[generative gestaltung](#)  
[predray](#)  
[fractaltables](#)  
[typestar](#)  
[3D SuperShapes](#)

## About

controlP5 is a library written by [Andreas Schlegel](#) for the programming environment [processing](#). Last update, 07/30/2015.

Controllers to build a graphical user interface on top of your processing sketch include Sliders, Buttons, Toggles, Knobs, Textfields, RadioButtons, Checkboxes amongst others and can be easily added to a processing sketch. They can be arranged in separate control PGraphics contexts, and can be organized in tabs or groups. → [read more](#).

## Installation

Unzip and put the extracted controlP5 folder into the libraries folder of your processing sketches. Reference and examples are included in the controlP5 folder.

## Examples

Find a list of examples in the current distribution of controlP5, or have a look by following the links below. If you want to share any examples, please let me know (andi at sojamo dot de).

[controllers](#)

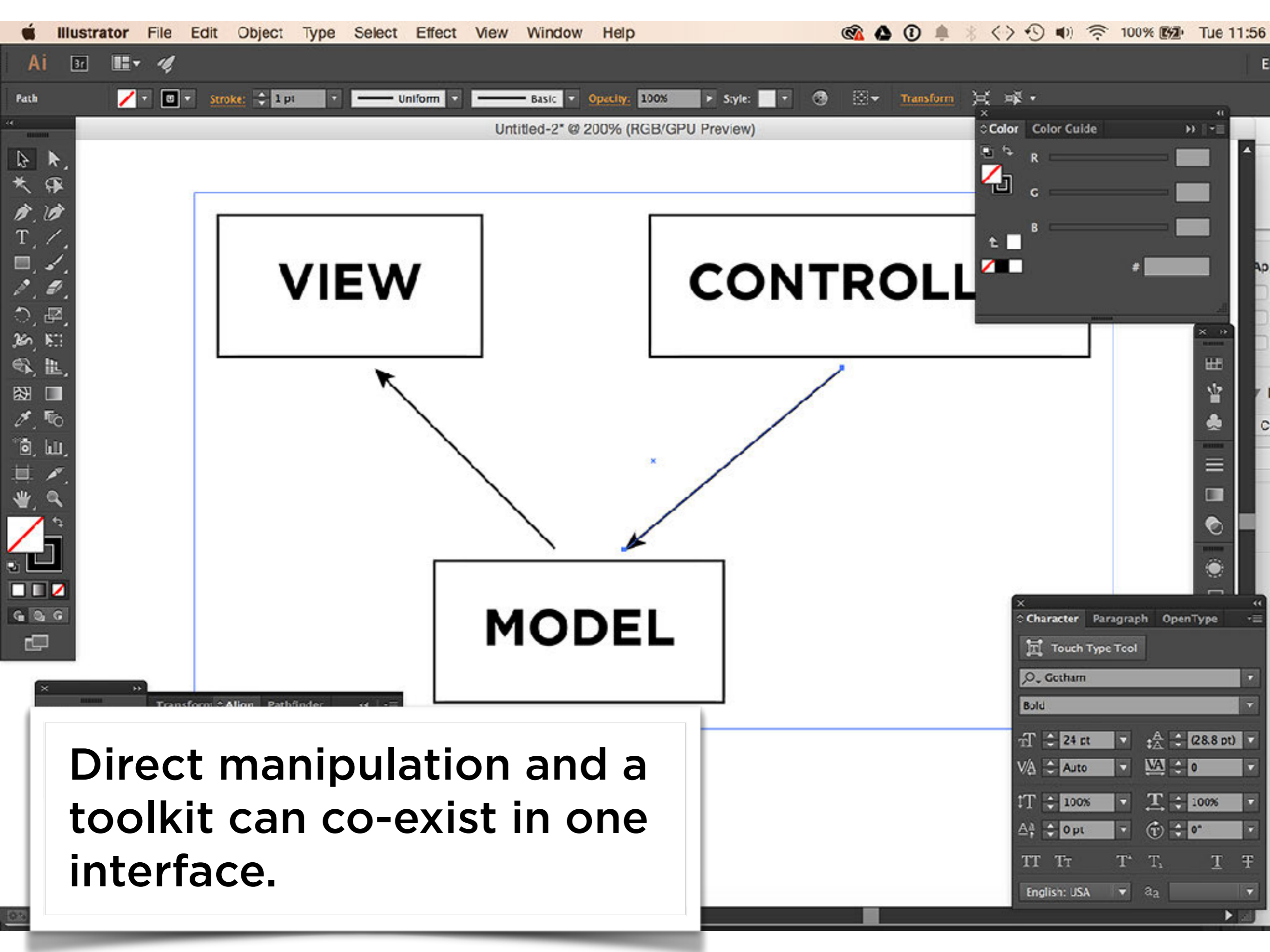
[controllers/ControlP5accordion](#)

[controllers/ControlP5bang](#)

[controllers/ControlP5button](#)

[controllers/ControlP5canvas](#)

[controllers/ControlP5chart](#)



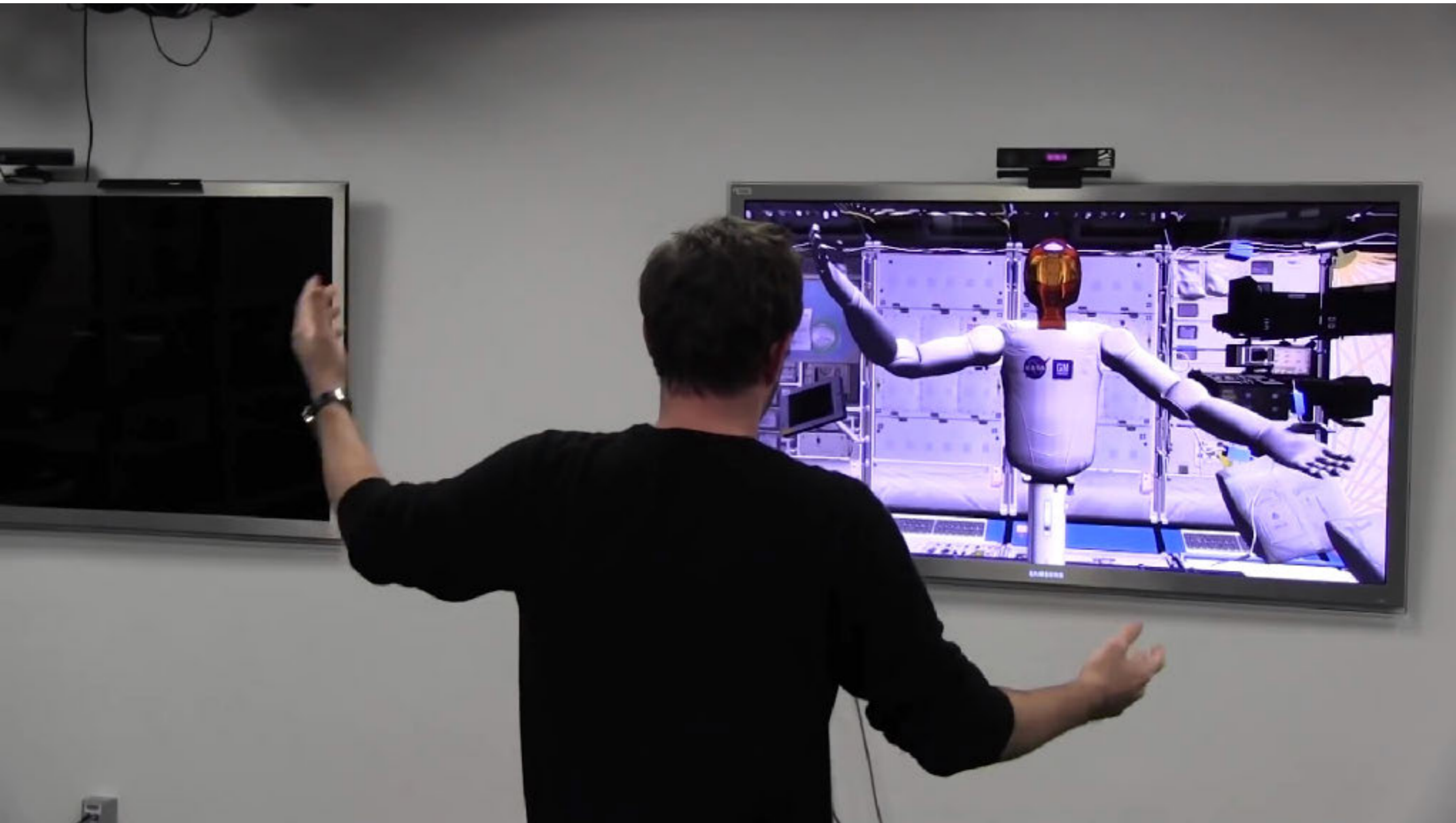
**VIEW**

**CONTROLL**

**MODEL**

**Direct manipulation and a toolkit can co-exist in one interface.**

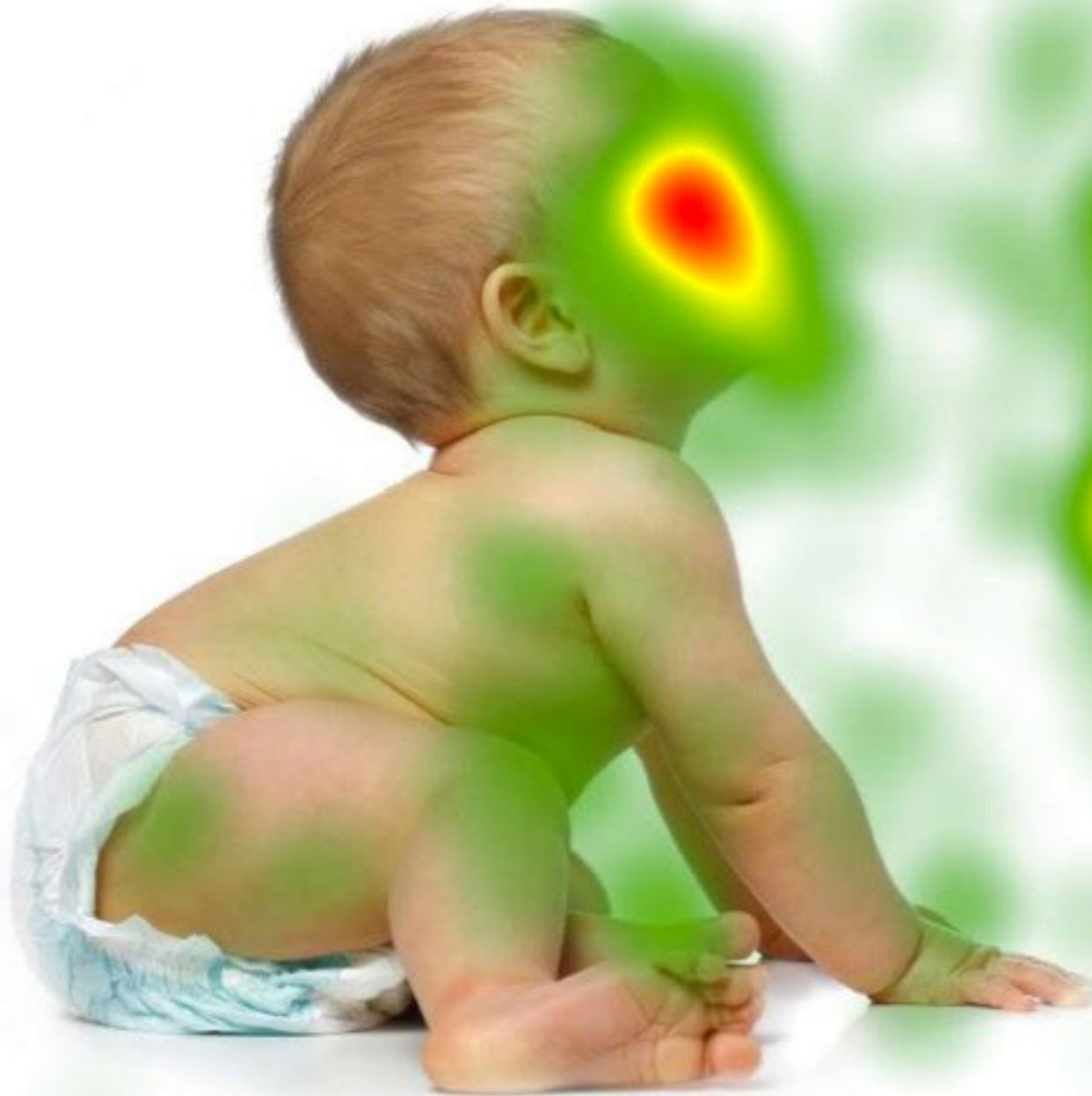
# Kinect



**NASA / Jet propulsion laboratory**



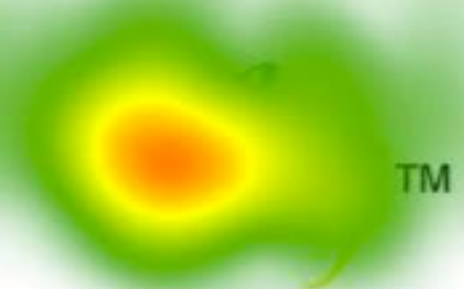
# Eye tracking



Eye tracking data for the  
most sensitive skin.

...to add the...leals and moisture  
...ive d...eal.

...nique high-absorbency natural-blend cotton  
...provides cotton-soft, extra thick, gel-free protection  
...baby's sensitive skin. The chlorine-free materials and  
...polymers is non-toxic and non-irritating. Clinically  
...matrician recommended for babies with allergies  
and sensitive skin.



TM

If you are not satisfied with the baby leakage protection, you will get your money back. Read more about our leakfree guarantee at [www.baby.com](http://www.baby.com)

# Myo Armband



# On-world interfaces

