Leibniz Universität Hannover

Mensch-Computer-Interaktion 2

Scene Graphs

Human-Computer Interaction Group

Prof. Dr. Michael Rohs
michael.rohs@hci.uni-hannover.de

Lectures

Session	Date	Topic	
1	6.4.	Introduction	
2	13.4.	Interaction elements	GUI toolkits, interaction techniques
3	20.4.	Event handling	
4	27.4.	Scene graphs	
5	4.5.	Interaction techniques	
	11.5.	no class (CHI)	
	18.5.	no class (spring break)	
6	25.5.	Experiments	design and analysis of experiments
7	1.6.	Data Analysis	
8	8.6.	Data Analysis	
9	15.6.	Visualization	
10	22.6.	Visualization	current topics beyond-desktop UIs
11	29.6.	Modeling interaction	
12	6.7.	Computer vision for interaction	
13	13.7.	Computer vision for interaction	

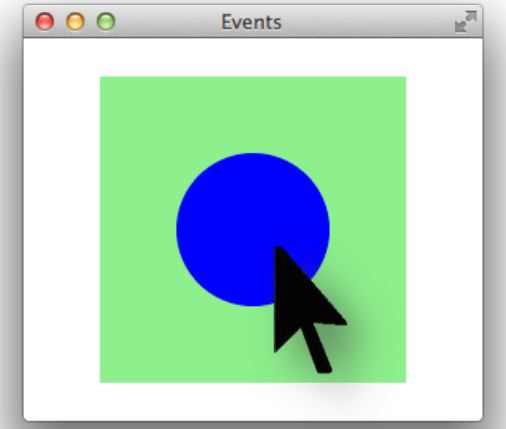
Klausur:
28.7.2016
8-11 Uhr
HG E214

EVENT DELIVERY IN THE SCENE GRAPH

Mouse and Key Events on Nodes of the Scene Graph

- 3 nodes in scene graph: A blue circle in front of a green rectangle in front of a white background:

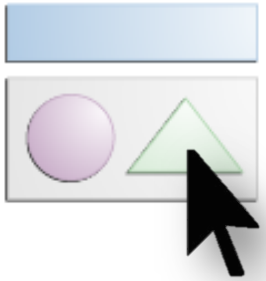
```
StackPane root = new StackPane();
Rectangle r = new Rectangle(200, 200, Color.GREEN);
Circle c = new Circle(50, Color.BLUE);
root.getChildren().addAll(r,c);
```



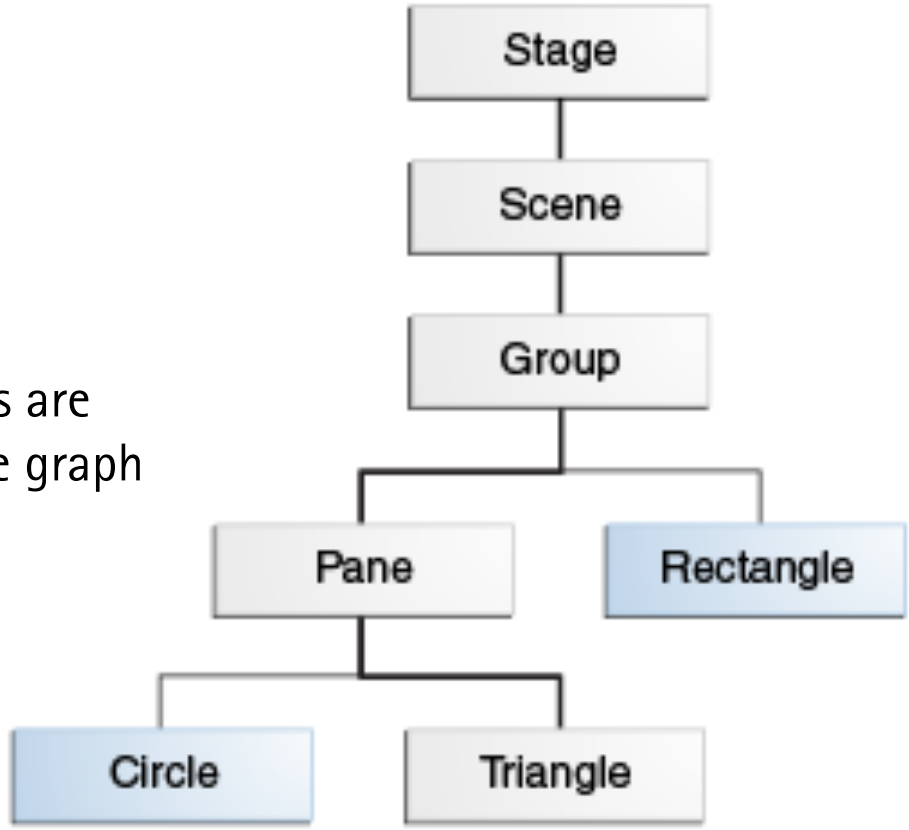
- System performs event transformation
  - "Click occurred at (x,y)" → "Click occurred on blue circle"
- Mouse events on a node
  - MousePressed, MouseReleased, MouseClicked, etc.
- Key events on the node with focus
  - KeyPressed, KeyReleased, KeyTyped

Example Scene Graph

Scene



Scene Graph

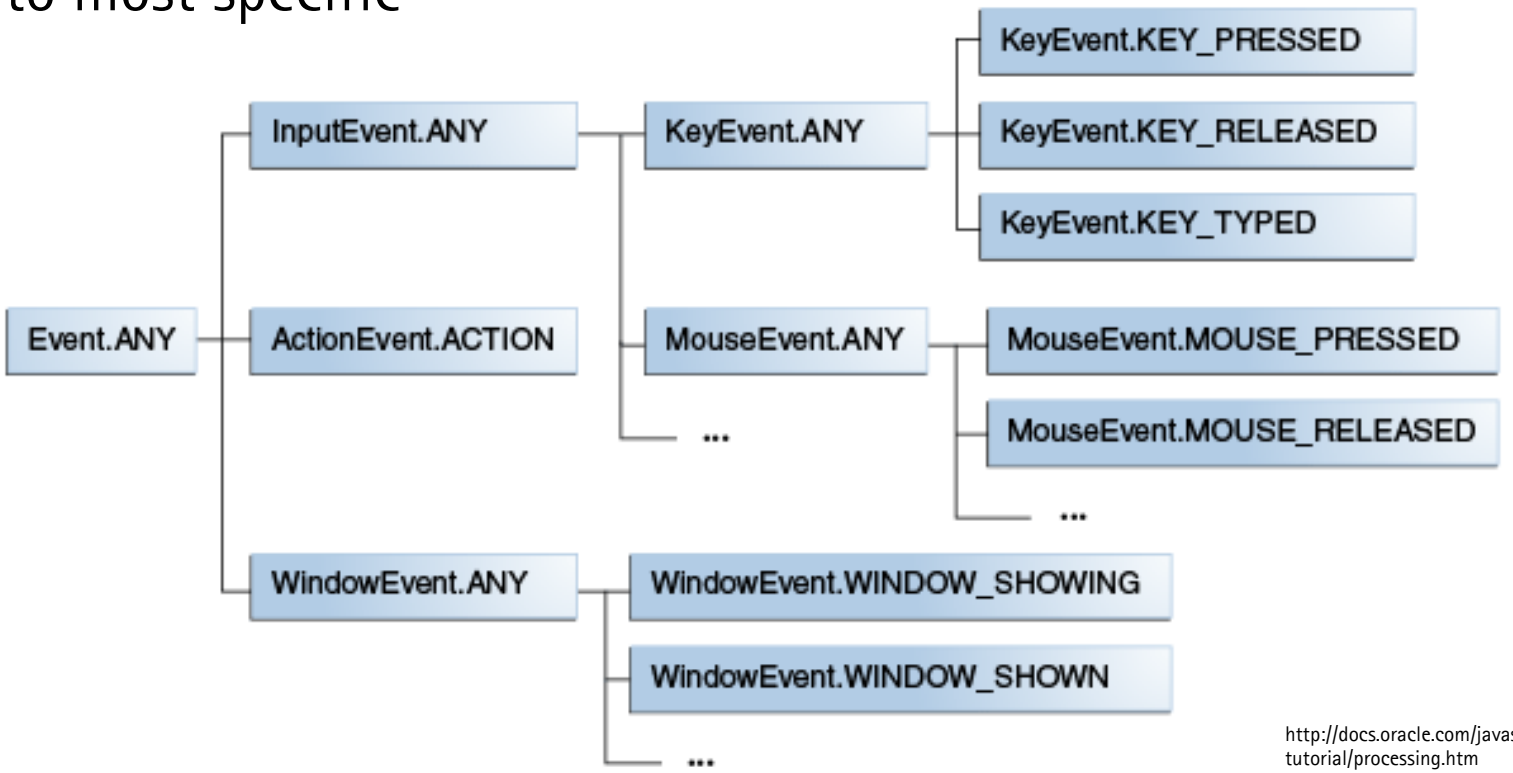


Event delivery: Events of different types are delivered to different nodes of the scene graph

http://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

Event Types

- Event: Occurrence of something of interest
- Event types: Hierarchy from most general (Event.ANY) to most specific



http://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

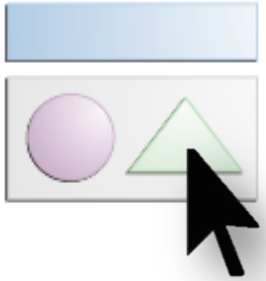
Event Source and Target

- Source: Origin of event
  - Reflects current position in the event dispatch chain
- Target: Node on which action occurred
  - Any node in the scene graph may be an event target
- Event filters and handlers
  - Each node may have event filters and handlers to process events
  - Each handler/filter processes events of a specific type
- Event filters
  - Down the event dispatch chain: event capturing phase
- Event handlers
  - Up the event dispatch chain: event bubbling phase

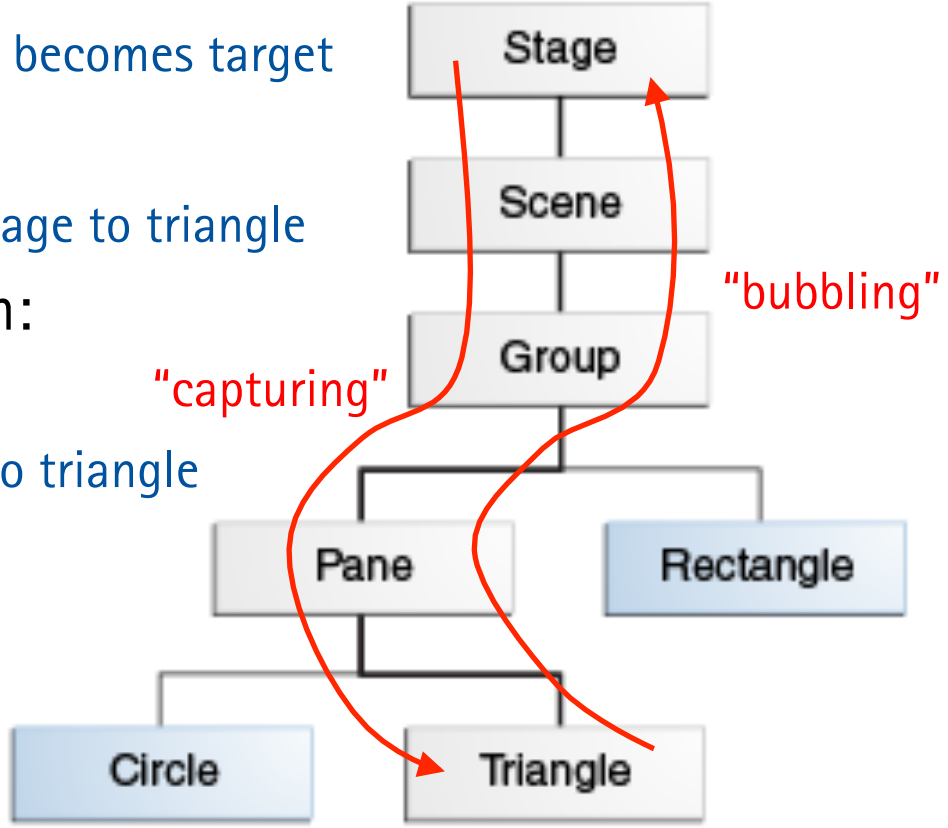
Event Delivery

- Target selection
  - Mouse events: Node at the cursor location
  - Key events: Node that has the focus (scene has the focus by default)
- Route construction
  - Node's buildEventDispatchChain creates chain of dispatchers
    - Default implementation follows layout hierarchy
    - Target specifies event dispatch chain
  - Route may be modified (rarely necessary)
  - Events may be modified or consumed during delivery

Event Delivery Example



- Target selection
  - Mouse click on triangle: triangle becomes target
- Route construction
  - Triangle's dispatch chain from stage to triangle
- Down the event dispatch chain: Event capturing phase
  - MouseEvent travels from stage to triangle
  - Event filters invoked
- Up the event dispatch chain: Event bubbling phase
  - From the triangle to the stage
  - Event handlers invoked

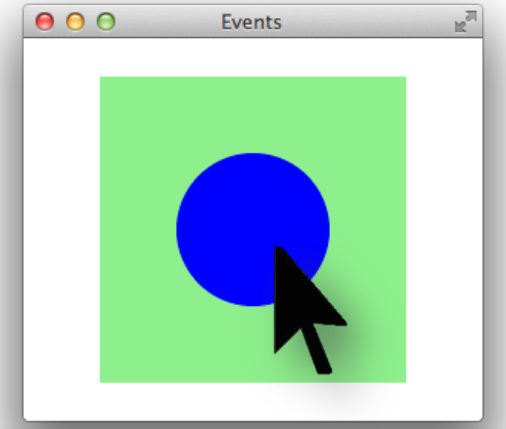


http://docs.oracle.com/javase/8/javafx/events-tutorial/processing.htm

Event Filters and Handlers

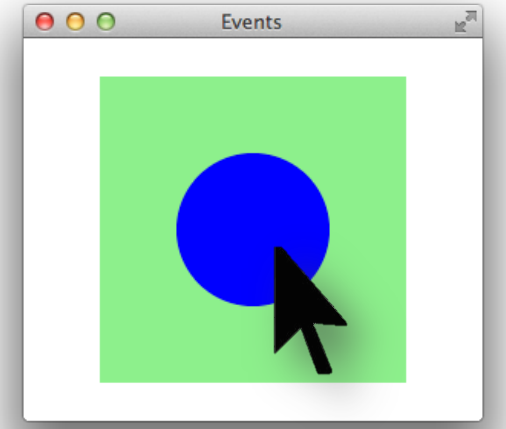
```
Circle circle = new Circle(50, Color.BLUE);

circle.addEventFilter(MouseEvent.MOUSE_PRESSED, e -> {
    System.out.println("\ncircle, event filter:");
    System.out.println(e);
    // e.consume();
});

circle.addEventHandler(MouseEvent.MOUSE_PRESSED, e -> {
    System.out.println("\ncircle, event handler:");
    System.out.println(e);
});
```

consume stops further event delivery



Event Filters and Handlers

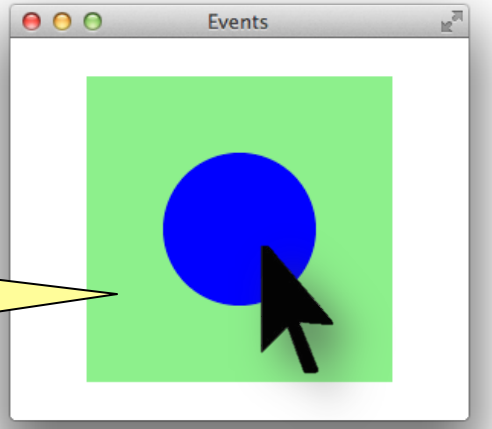
```
StackPane root = new StackPane();
Rectangle r = new Rectangle(200, 200, Color.LIGHTGREEN);
Circle c = new Circle(50, Color.BLUE);
root.getChildren().addAll(r,c);
```




```
root.addEventFilter(MouseEvent.MOUSE_PRESSED, e -> {...});
root.addEventHandler(MouseEvent.MOUSE_PRESSED, e -> {...});
c.addEventFilter(MouseEvent.MOUSE_PRESSED, e -> {...});
c.addEventHandler(MouseEvent.MOUSE_PRESSED, e -> {...});
r.addEventFilter(MouseEvent.MOUSE_PRESSED, e -> {...});
r.addEventHandler(MouseEvent.MOUSE_PRESSED, e -> {...});
```

each event filter/handler here just prints the event

Event Filters and Handlers, Output



rectangle's event filter not invoked

time

Pane, event filter:

MouseEvent [source = StackPane, target = Circle, x = 162.0, y = 130.0, button = PRIMARY]

Circle, event filter:

source changes

MouseEvent [source = Circle, target = Circle, x = 12.0, y = 5.0, button = PRIMARY]

coordinates converted to circle's local coordinate system

Circle, event handler:

MouseEvent [source = Circle, target = Circle, x = 12.0, y = 5.0, button = PRIMARY]

Pane, event handler:

MouseEvent [source = StackPane, target = Circle, x = 162.0, y = 130.0, button = PRIMARY]

Convenience Methods

- Convenience methods in class Node to simplify event handling
- Pattern:

```
setOn{event type}(EventHandler
        <? super {event class}> h)
```

- Example:

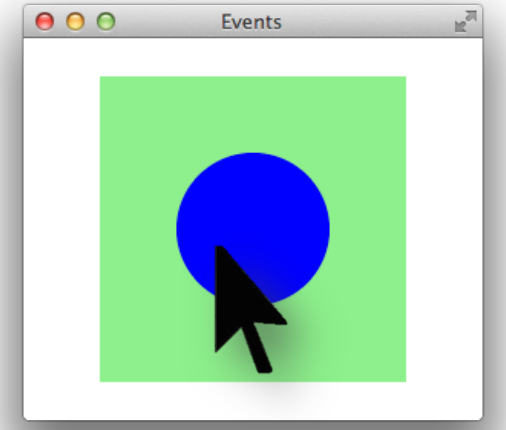
```
r.setOnMousePressed(
        (MouseEvent e) -> { ... });
```

lambda expression

setOnX	Event Type
KeyPressed	KeyEvent
KeyReleased	KeyEvent
KeyTyped	KeyEvent
MousePressed	MouseEvent
MouseReleased	MouseEvent
MouseClicked	MouseEvent
MouseMoved	MouseEvent
MouseEntered	MouseEvent
MouseExited	MouseEvent
MouseDragEntered	MouseDragEvent
MouseDragExited	MouseDragEvent
MouseDragOver	MouseDragEvent
MouseDragReleased	MouseDragEvent
MouseDragged	MouseEvent
Action	ActionEvent
...	ScrollEvent

Coordinate Systems for Mouse Events

- Scene graph has multiple coordinate systems
  - Each node has a local coordinate system
- x, y
  - Coordinate system of the MouseEvent's source node
- sceneX, sceneY
  - Coordinate system of the node's scene
- screenX, screenY
  - Screen coordinate system



x = -7.0
sceneX = 143.0
screenX = 713.0

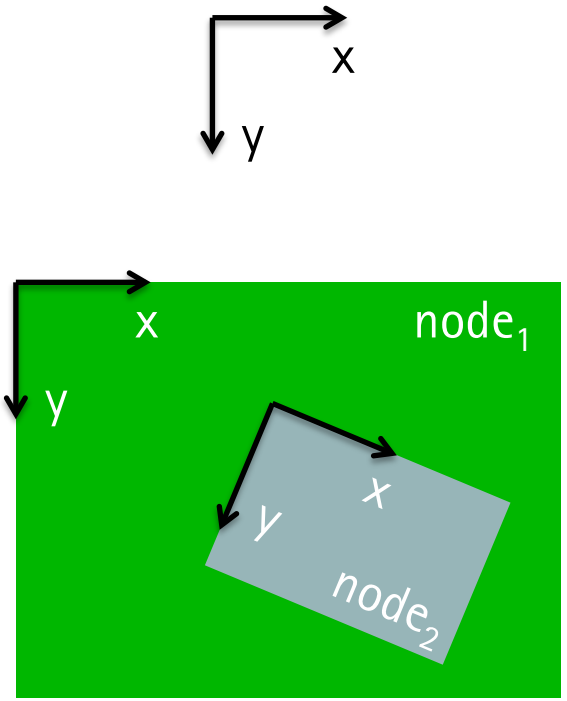
COORDINATE SYSTEMS

Coordinate Values

- Coordinates are double values
  - Integer coordinates refer to pixel corners
  - Midpoints between integer coordinates refer to pixel centers
- Example:
  - Stage coordinates (0.5, 0.5) refer to center of upper left pixel on Stage
- Example:
  - Rectangle at (0, 0) with size 10 by 10 pixels
  - Extent: Upper left corner of upper left pixel on Stage to lower right corner of 10th pixel on 10th scanline
  - Center of first pixel inside rectangle: (0.5, 0.5)
  - Center of last pixel inside rectangle: (9.5, 9.5)

Node Coordinate Systems

- Each node has its own coordinate system
  - x axis to right, y axis downwards
  - May be translated, rotated, scaled, and sheared w.r.t. parent node's coordinate system
- Shape-nodes define their geometry within local coordinate system
  - Rectangle: x, y, width, height
  - Circle: centerX, centerY, radius



Affine Transformations

- Modify a node's local coordinate system
- Translation
  - Shifts the origin of the coordinate system along x or y axis
- Rotation
  - Rotates the coordinate system about a "pivot" point
  - Pivot point is center of layout bounds by default
- Scaling
  - Scales the axes of the coordinate system about a "pivot" point
  - Pivot point is center of layout bounds by default
- Shearing
  - Rotates the axes, such that they are no longer perpendicular

Affine Transformations

- Affine transformations are linear mappings of coordinates
  - Preserve straightness and parallelism of lines
- Represented as matrixes
- Transformations typically along 3 axes for generality
- Typically implemented in hardware (graphics card)
- Transformations may be chained

Bounding Rectangles of Node

- boundsInLocal
  - Bounding rectangle in untransformed local coordinates
  - Includes the Node's shape geometry, stroke, effects, clip, but no transforms
- boundsInParent
  - Bounding rectangle after all transformations in parent's coordinate system
  - Includes shape geometry, stroke, effects, clip, and transforms
  - scaleX/scaleY, rotate, translateX/translateY, layoutX/layoutY
- layoutBounds
  - Bounding rectangle for layout calculations, local coordinates
  - Includes shape geometry and stroke, but no effects, clip, or transforms
  - For resizable nodes (Regions/Controls) layoutBounds is 0,0 width x height

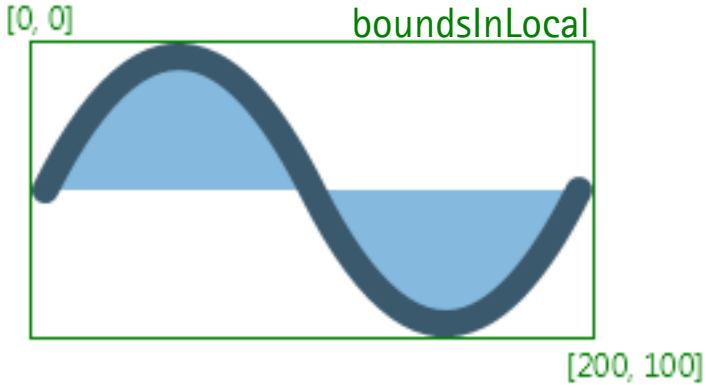
Example: Bounding Rectangles of Node

Smallest axis-parallel enclosing rectangle in the respective coordinate system

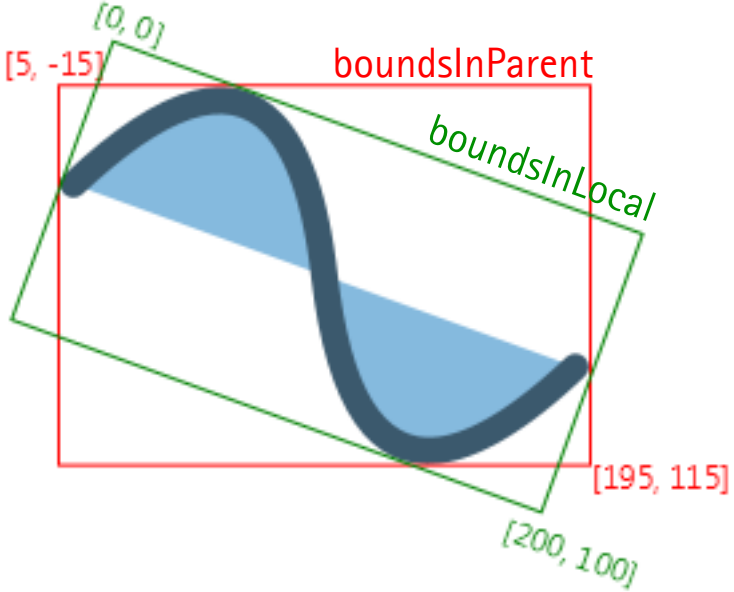
Green: boundsInLocal

Red: boundsInParent

original



rotated by 20°



http://docs.oracle.com/javase/8/javafx/api/javafx/scene/Node.html

Example: Bounding Rectangles of Node

Local bounds and parent bounds include stroke, stroke is centered by default

[x:10.0 y:10.0
width:100.0 height:100.0
strokeWidth:0]

Bounds: [x:10.0 y:10.0
width:100.0 height:100.0]

(red rectangle shows bounding rectangle)

[x:10.0 y:10.0
width:100.0 height:100.0
strokeWidth:5]

Bounds: [x:7.5 y:7.5
width:105 height:105]

http://docs.oracle.com/javase/8/javafx/api/javafx/scene/Node.html

Local Bounds vs. Layout Bounds

- Without effects

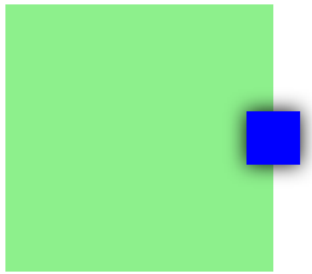
```
Rectangle rect = new Rectangle(100, 100, Color.LIGHTGREEN);
Rectangle handle = new Rectangle(20, 20, Color.BLUE);
handle.setTranslateX(90);
handle.setTranslateY(40);
Group group = new Group(rect, handle);
```

handle	minX	minY	maxX	maxY
getBoundsInLocal()	0.0	0.0	20.0	20.0
getBoundsInParent()	90.0	40.0	110.0	60.0
getLayoutBounds()	0.0	0.0	20.0	20.0

Local Bounds vs. Layout Bounds

- With effect (drop shadow)

```
Rectangle rect = new Rectangle(100, 100, Color.LIGHTGREEN);
Rectangle handle = new Rectangle(20, 20, Color.BLUE);
handle.setTranslateX(90);
handle.setTranslateY(40);
handle.setEffect(new DropShadow(10, Color.BLACK));
Group group = new Group(rect, handle);
```



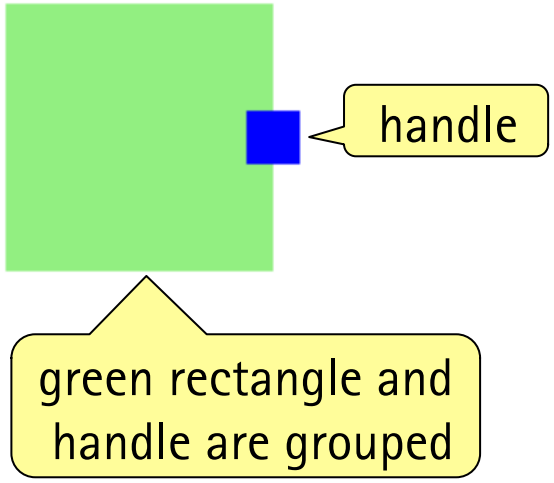
handle	minX	minY	maxX	maxY
getBoundsInLocal()	-9.0	-9.0	29.0	29.0
getBoundsInParent()	81.0	31.0	119.0	69.0
getLayoutBounds()	0.0	0.0	20.0	20.0

Transformations between Coordinate Systems

- localToParentTransform
  - Transforming coordinates in local coordinate system to coordinates in parent's coordinate system
- localToSceneTransform
  - Transforming coordinates in local coordinate system to coordinates in scene's coordinate system
- Transform (class)
  - Represents an affine transformation, i.e. a 3x4 matrix (for 3D transformations)
  - Invert transforms: createInverse() (exception if none exists)
  - Concatenate transforms: createConcatenation(Transform transform)

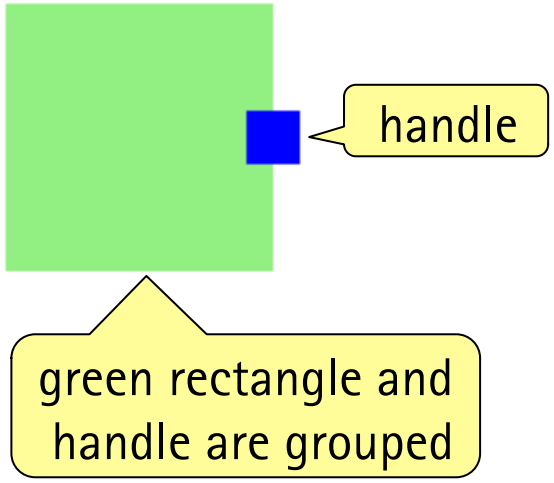
Example: A Moveable and Width-Resizable Rectangle

```
Pane root = new Pane();
Rectangle rect = new Rectangle(100, 100, Color.LIGHTGREEN);
Rectangle handle = new Rectangle(20, 20, Color.BLUE);
handle.setTranslateX(rect.getWidth() - handle.getWidth() / 2);
handle.setTranslateY(rect.getHeight() / 2 - handle.getHeight() / 2);

Group group = new Group(rect, handle);
root.getChildren().add(group);
Scene scene = new Scene(root, 800, 400);

handle.setCursor(Cursor.H_RESIZE);
rect.setCursor(Cursor.OPEN_HAND);
```



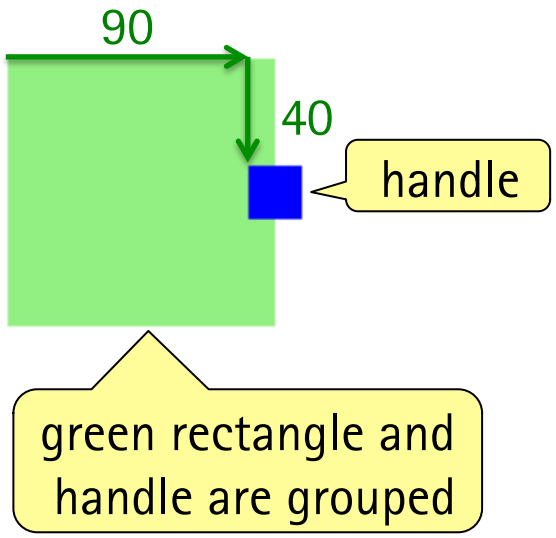
Example: A Moveable and Width-Resizable Rectangle

```
Pane root = new Pane();
Rectangle rect = new Rectangle(100, 100, Color.LIGHTGREEN);
Rectangle handle = new Rectangle(20, 20, Color.BLUE);
handle.setTranslateX(100 - 20 / 2);
handle.setTranslateY(100 / 2 - 20 / 2);

Group group = new Group(rect, handle);
root.getChildren().add(group);
Scene scene = new Scene(root, 800, 400);

handle.setCursor(Cursor.H_RESIZE);
rect.setCursor(Cursor.OPEN_HAND);
```

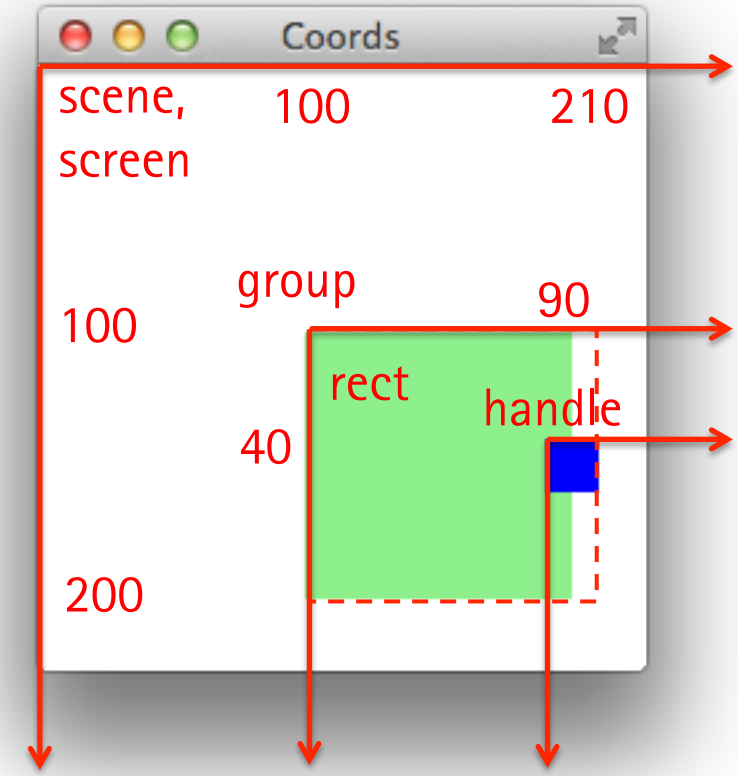


Example: A Moveable and Width-Resizable Rectangle

```
Pane root = new Pane();
Rectangle rect = new Rectangle(100, 100, Color.LIGHTGREEN);
Rectangle handle = new Rectangle(20, 20, Color.BLUE);
handle.setTranslateX(90); // shift handle's coordinate system 90 pixels right
handle.setTranslateY(40); // shift handle's coordinate system 40 pixels down

Group group = new Group(rect, handle);
root.getChildren().add(group);
Scene scene = new Scene(root, 800, 400);

handle.setCursor(Cursor.H_RESIZE);
rect.setCursor(Cursor.OPEN_HAND);
```



Example: A Moveable and Width-Resizable Rectangle



BoundingBoxes:

group in local:
[minX:0, minY:0, maxX:110, maxY:100, width:110, height:100]

group in parent:
minX:100, minY:100, maxX:210, maxY:200, width:110, height:100]

rect in local:
[minX:0, minY:0, maxX:100, maxY:100, width:100, height:100]

rect in parent:
[minX:0, minY:0, maxX:100, maxY:100, width:100, height:100]

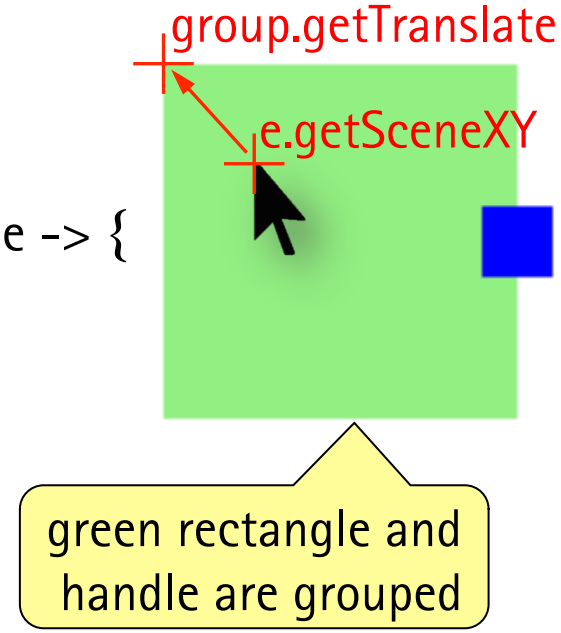
handle in local:
[minX:0, minY:0, maxX:20, maxY:20, width:20, height:20]

handle in parent:
[minX:90, minY:40, maxX:110, maxY:60, width:20, height:20]

Example: A Moveable and Width-Resizable Rectangle

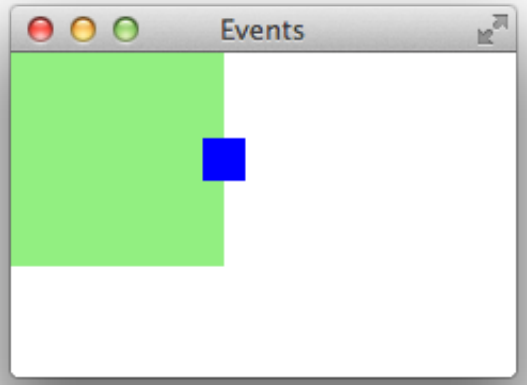
Drag group (green rectangle and handle):

```
group.addEventHandler(MouseEvent.MOUSE_PRESSED, e -> {
    offsetX = group.getTranslateX() - e.getSceneX();
    offsetY = group.getTranslateY() - e.getSceneY();
});

group.addEventHandler(MouseEvent.MOUSE_DRAGGED, e -> {
    group.setTranslateX(offsetX + e.getSceneX());
    group.setTranslateY(offsetY + e.getSceneY());
});
```

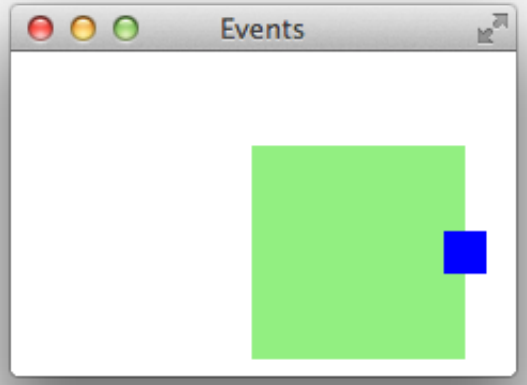


Example: A Moveable and Width-Resizable Rectangle

- Before dragging:
  - group: layoutX = 0, translateX = 0
  - rect: layoutX = 0, translateX = 0, x = 0
  - handle: layoutX = 0, translateX = 90, x = 0



- After dragging:
  - group: layoutX = 0, **translateX = 113**
  - rect: layoutX = 0, translateX = 0, x = 0
  - handle: layoutX = 0, translateX = 90, x = 0



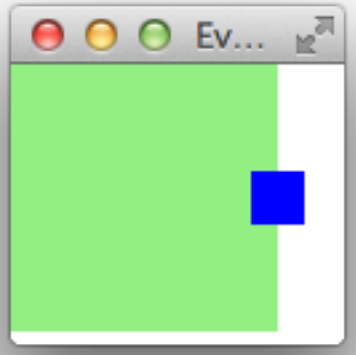
Example: A Moveable and Width-Resizable Rectangle

Drag handle (to resize):

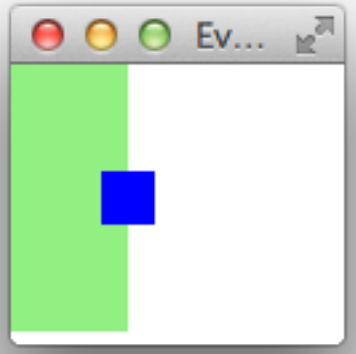
```
handle.addEventHandler(MouseEvent.MOUSE_PRESSED, e -> {
    offsetX = handle.getTranslateX() - e.getSceneX();
    offsetY = handle.getTranslateY() - e.getSceneY();
    e.consume(); // prevent further processing
});
handle.addEventHandler(MouseEvent.MOUSE_DRAGGED, e -> {
    double handleHalfWidth = handle.getWidth() / 2;
    double newTranslateX = Math.max(-handleHalfWidth, offsetX + e.getSceneX());
    double newWidth = newTranslateX + handleHalfWidth;
    handle.setTranslateX(newTranslateX);
    rect.setWidth(newWidth);
    e.consume(); // prevent further processing
});
```

Example: A Moveable and Width-Resizable Rectangle

- Before resizing:
  - group, bounds in local: minX = 0, maxX = 110
  - group, bounds in parent: minX = 0, maxX = 110
  - rect, bounds in local: minX = 0, maxX = 100
  - rect, bounds in parent: minX = 0, maxX = 100
  - handle, bounds in local: minX = 0, maxX = 20
  - handle, bounds in parent: minX = 90, maxX = 110

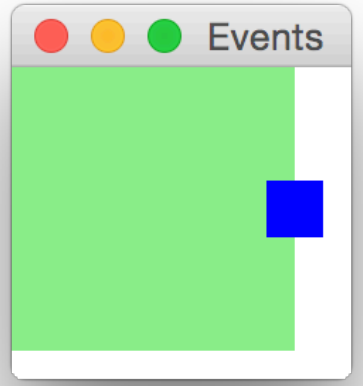


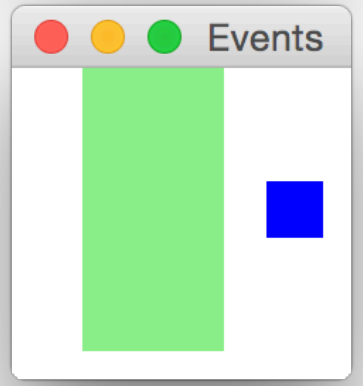
- After resizing:
  - group, bounds in local: minX = 0, maxX = 54
  - group, bounds in parent: minX = 0, maxX = 54
  - rect, bounds in local: minX = 0, maxX = 44
  - rect, bounds in parent: minX = 0, maxX = 44
  - handle, bounds in local: minX = 0, maxX = 20
  - handle, bounds in parent: minX = 34, maxX = 54



Scaling a Shape Node (Rectangle, Circle, etc.)

- Before scaling:
  - group, bounds in local: minX = 0, maxX = 110
  - group, bounds in parent: minX = 0, maxX = 110
  - rect, bounds in local: minX = 0, maxX = 100
  - rect, bounds in parent: minX = 0, maxX = 100
- After scaling (rect.setScaleX(0.5)):
  - group, bounds in local: minX = 25, maxX = 110
  - group, bounds in parent: minX = 25, maxX = 110
  - rect, bounds in local: minX = 0, maxX = 100
  - rect, bounds in parent: minX = 25, maxX = 75

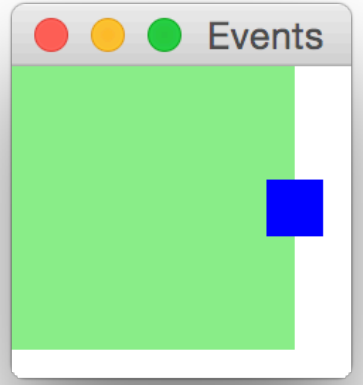


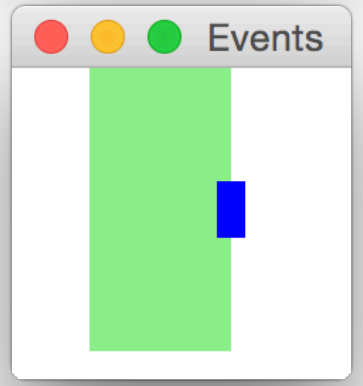


scales around center,
bounds in local stay,
bounds in parent change

Scaling a Group Node

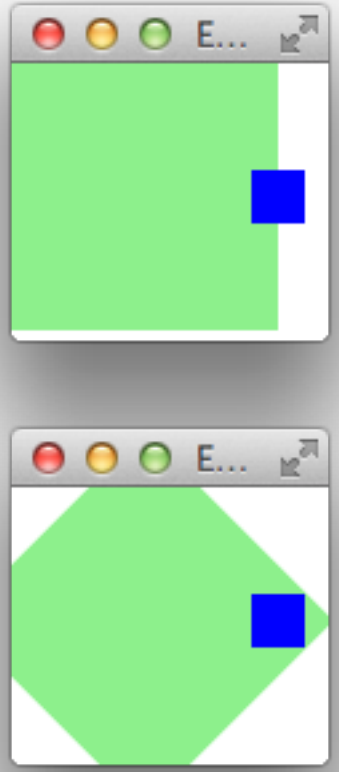
- Before scaling:
  - group, bounds in local: minX = 0, maxX = 110
  - group, bounds in parent: minX = 0, maxX = 110
  - rect, bounds in local: minX = 0, maxX = 100
  - rect, bounds in parent: minX = 0, maxX = 100
  - handle, bounds in local: minX = 0, maxX = 20
  - handle, bounds in parent: minX = 90, maxX = 110
- After scaling (group.setScaleX(0.5)):
  - group, bounds in local: minX = 0, maxX = 110
  - group, bounds in parent: minX = 27.5, maxX = 82.5
  - rect bounds: no change
  - handle bounds: no change





Rotating a Shape Node

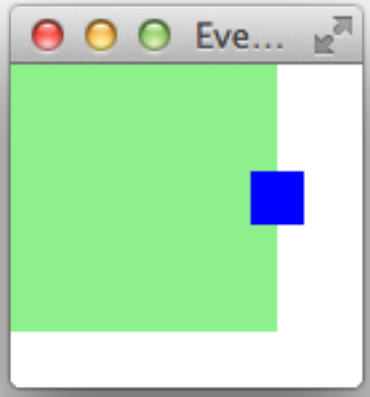
- Before rotating:
  - group, bounds in local: minX = 0, maxX = 110
  - group, bounds in parent: minX = 0, maxX = 110
  - rect, bounds in local: minX = 0, maxX = 100
  - rect, bounds in parent: minX = 0, maxX = 100
- After rotating (rect.setRotate(45)):
  - group, bounds in local: minX = -20.7, maxX = 120.7
  - group, bounds in parent: minX = -20.7, maxX = 120.7
  - rect, bounds in local: minX = 0, maxX = 100
  - rect, bounds in parent: minX = -20.7, maxX = 120.7

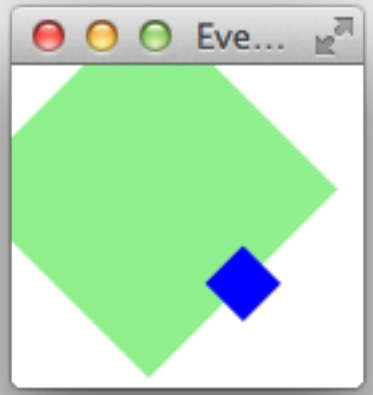


rotates around center, bounds in local stay, bounds in parent change

Rotating a Group Node

- Before rotating:
  - group, bounds in local: minX = 0, maxX = 110
  - group, bounds in parent: minX = 0, maxX = 110
  - rect, bounds in local: minX = 0, maxX = 100
  - rect, bounds in parent: minX = 0, maxX = 100
  - handle, bounds in local: minX = 0, maxX = 20
  - handle, bounds in parent: minX = 90, maxX = 110
- After rotating (group.setRotate(45)):
  - group, bounds in local: minX = 0, maxX = 110
  - group, bounds in parent: minX = -19.2, maxX = 122.2
  - rect bounds: no change
  - handle bounds: no change



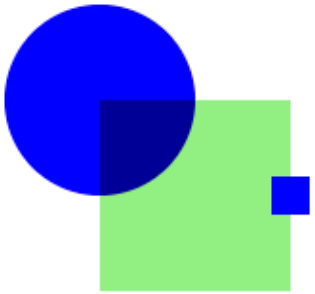


Leibniz Universität Hannover

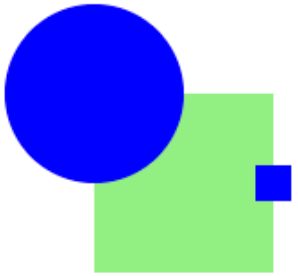
NODE PROPERTIES

Node Properties: Blend Mode

- Blend mode
  - How to blend the node into the scene behind it
  - Several different blend modes
  - circle.setBlendMode(BlendMode.MULTIPLY);

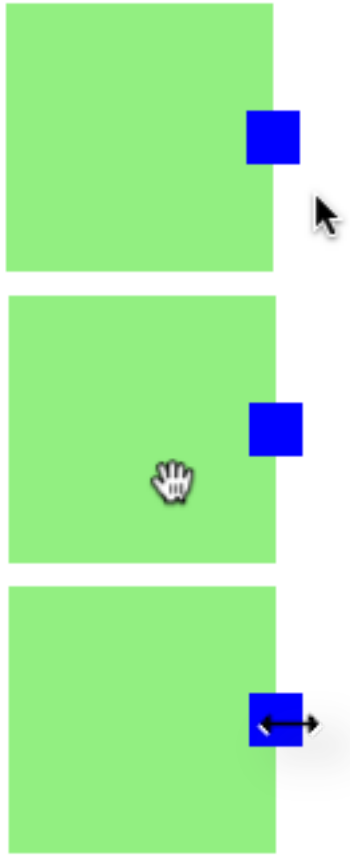


  - circle.setBlendMode(BlendMode.SRC_OVER);



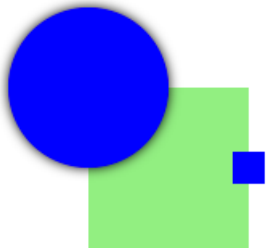
Node Properties: Cursor

- Cursor
  - The mouse cursor for this node
  - Helps user to discover interactive elements
  - rect.setCursor(Cursor.OPEN_HAND);
    - green moveable rectangle
  - rectHandle.setCursor(Cursor.H_RESIZE);
    - blue resize handle

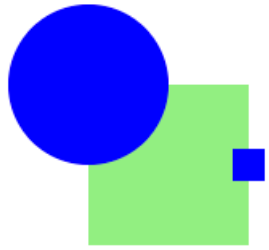


Node Properties: Effects

- Blurring, drop shadow, reflection, etc.



- circle.setEffect(new GaussianBlur(10));



no effect

- circle.setEffect(new DropShadow());

- circle.setEffect(new MotionBlur(-15, 30));

- circle.setEffect(new Reflection());

Other Node Properties

- Clipping
  - A node that defines the clipping shape of another node
  - Example: Circle shape for clipping an ImageView to round
- Opacity
  - Degree of transparency
- Parent
  - Parent in layout hierarchy
- Scene
  - The scene the node is part of

clipping image

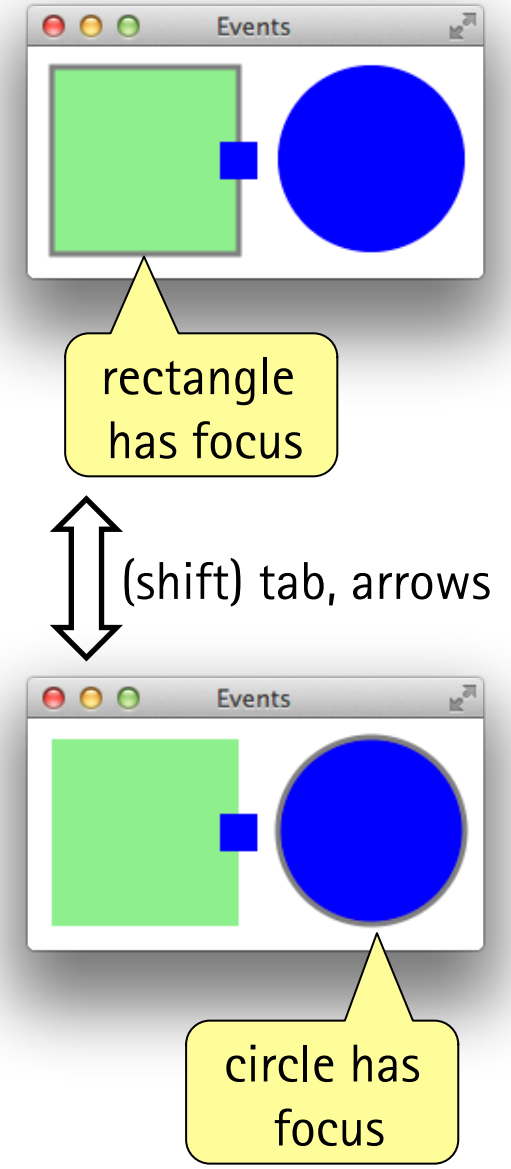
with ellipse

results in

Focused Nodes

- Nodes may be focus traversable

```
rect.setFocusTraversable(true);
circle.setFocusTraversable(true);

rect.focusedProperty().addListener((v,o,n) -> {
    rect.setStroke(Color.GRAY);
    rect.setStrokeWidth(n ? 3 : 0);
});

circle.focusedProperty().addListener((v,o,n) -> {
    circle.setStroke(Color.GRAY);
    circle.setStrokeWidth(n ? 3 : 0);
});
```



3D SHAPES

3D Shapes in JavaFX

- Scene graph nodes may have 3D geometry
  - Sphere, Box, Cylinder, MeshView
- 3D rendering is an optional feature
  - if (Platform.isSupported(ConditionalFeature.SCENE3D)) { ... }
- Camera position, direction, view frustum
- Lighting
- Material (surface) properties

3D Shapes Example

```
public void start(Stage stage) {
    Pane root = new Pane();
    Sphere s = new Sphere(40); // radius
    s.setTranslateX(100); s.setTranslateY(100);
    Box b = new Box(80, 80, 40); // width, height, depth
    b.setTranslateX(200); b.setTranslateY(100);
    Cylinder c = new Cylinder(40, 80); // radius, height
    c.setTranslateX(300); c.setTranslateY(100);
    root.getChildren().addAll(s, b, c);
    Scene scene = new Scene(root, 600, 200);
    stage.setTitle("3D Test");
    stage.setScene(scene);
    stage.show();
}
```

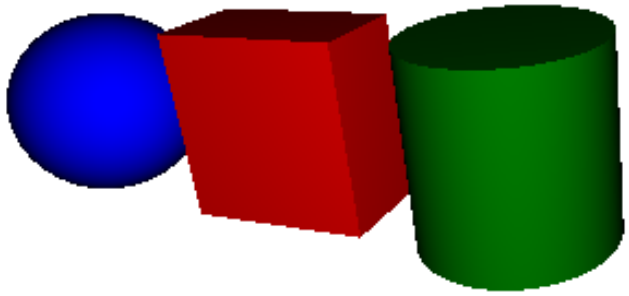
s.setDrawMode(DrawMode.FILL);

s.setDrawMode(DrawMode.LINE);

Viewing from a Specific Perspective

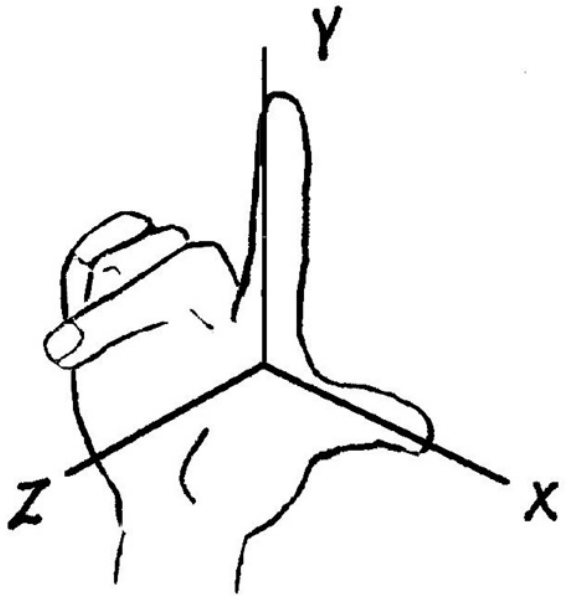
- Set perspective camera
- Set viewing direction
- Rotate about some axis
- Example

```
PerspectiveCamera camera = new PerspectiveCamera();
scene.setCamera(camera);
camera.setRotationAxis(new Point3D(1, 1, 0));
camera.setRotate(-30);
```



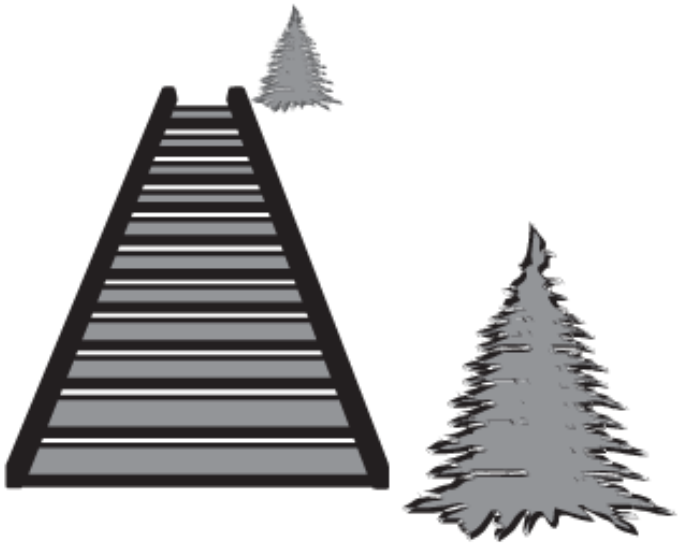
3D Coordinate System

- Right-hand coordinate system
- Mapping 3D scene to to 2D "screen"
  - Positive x axis: to the right
  - Positive y axis: down
  - Positive z axis: into the screen
- Mapping 3D to 2D: Projection



Orthographic vs. Perspective Projection

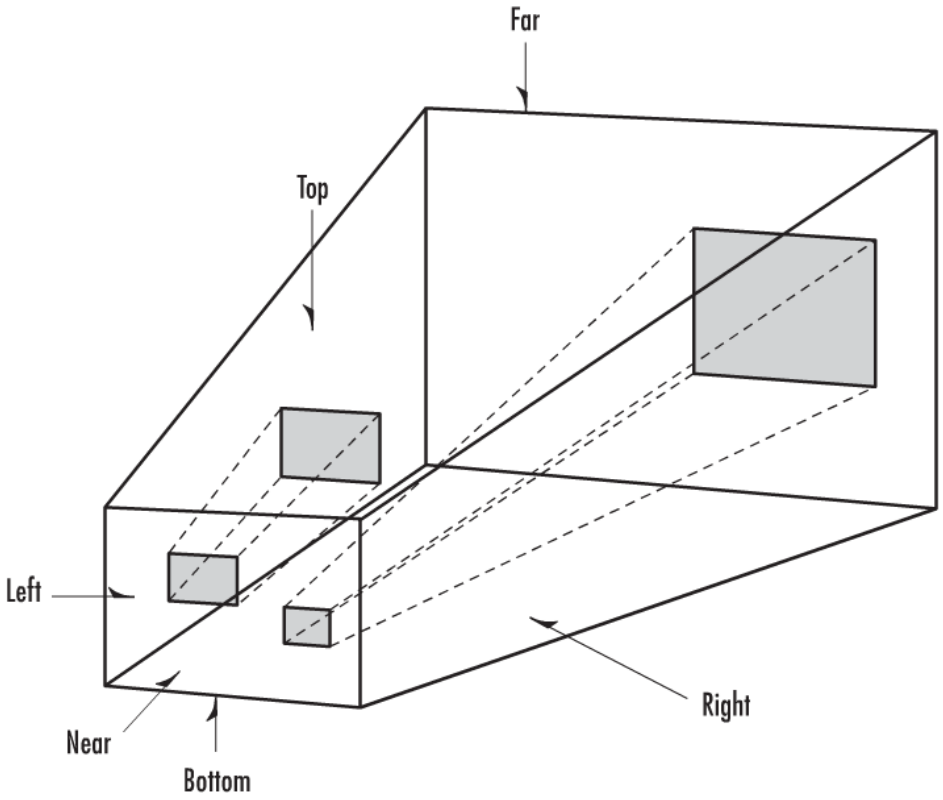
- Perspective projection
  - Distant objects appear smaller



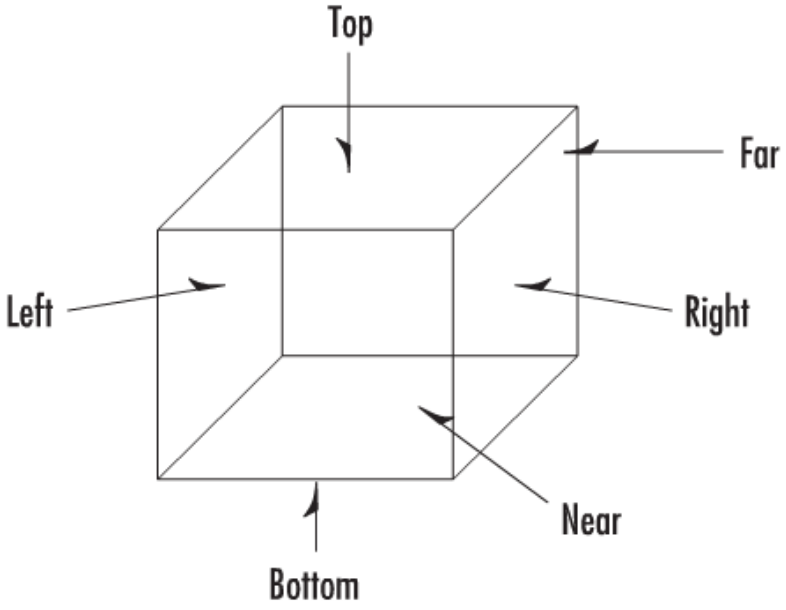
- Parallel projection
  - Distant objects same size

Camera View Concepts

- Perspective Projection
  - Distant objects appear smaller



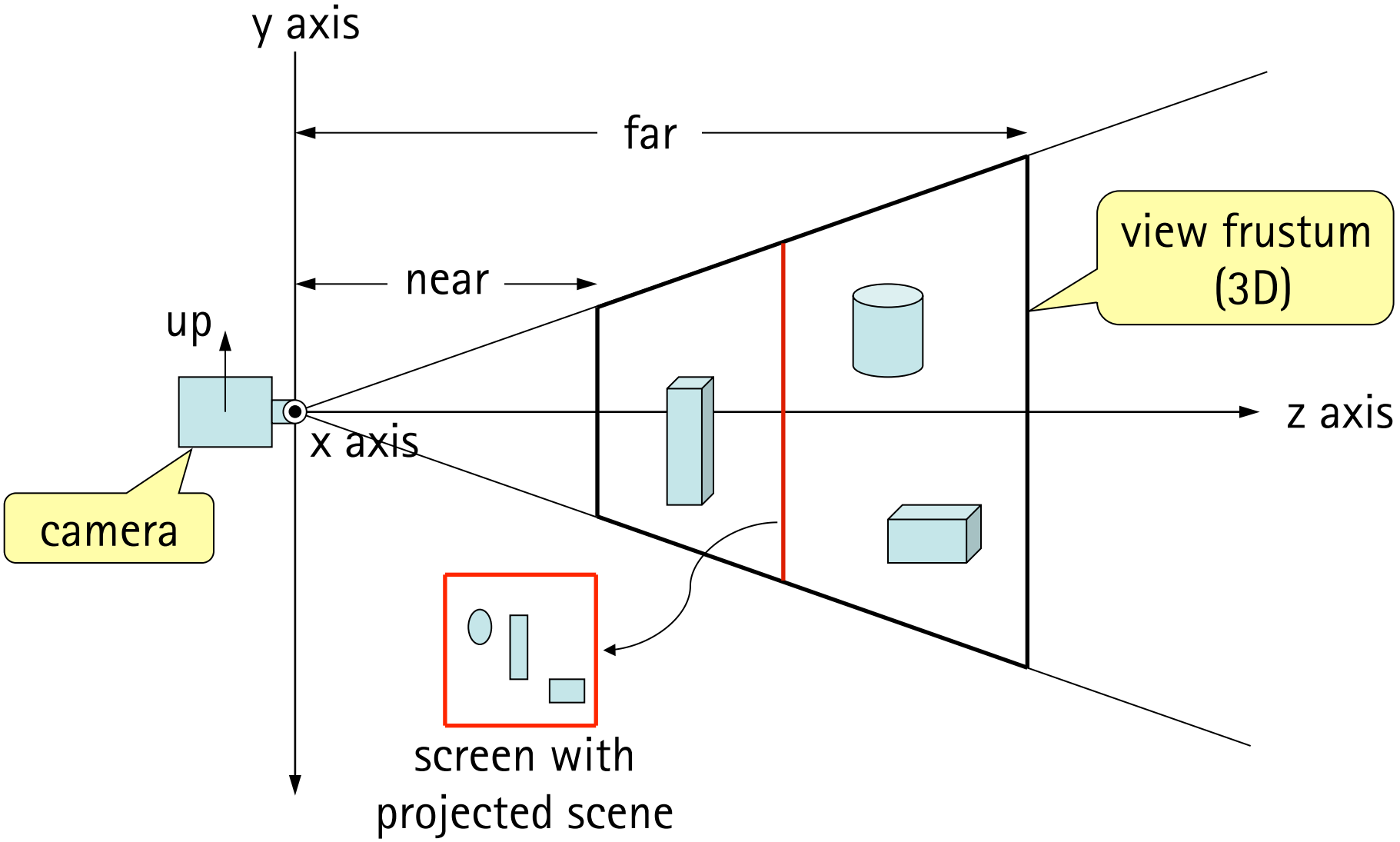
- Parallel Projection
  - Distant objects have same size



Perspective Camera

- 3D Scene is observed by a "camera"
- Parameters of perspective camera
  - Camera position: (0, 0, 0)
    - Origin of coordinate system
  - Viewing direction: (0, 0, 1)
    - Along positive z axis, negative y axis is up
  - Field of view: 30°
    - Opening angle of the camera
  - Near clipping distance
    - Closest distance the camera sees
  - Far clipping distance
    - Farthest distance the camera sees

Perspective Camera View Concepts

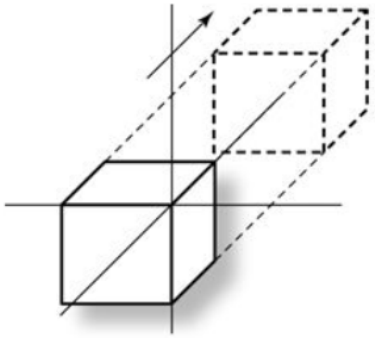


Perspective Camera in JavaFX

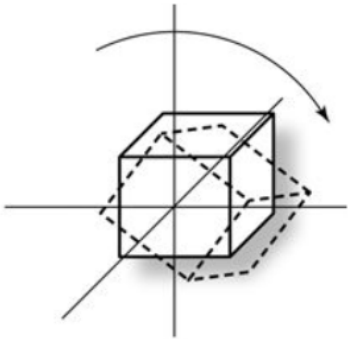
- Perspective camera is a Node object
  - May change position, orientation, opening angle, etc.
- Parameters of perspective camera
  - Camera position: (0, 0, 0)
  - Viewing direction: (0, 0, 1)
  - Field of view: 30°
  - Near clipping distance: 0.1
  - Far clipping distance: 100.0
  - Projection plane is at z = 0 (in scene coordinates)

Geometric Transformations in 3D

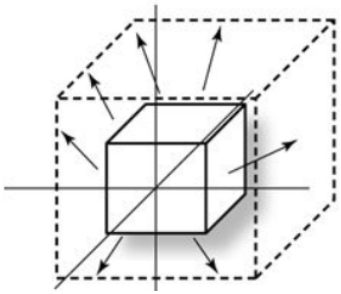
- Modeling transformations
  - Translating, rotating, scaling objects
- Viewing transformation
  - Specify location and orientation of viewer
  - Eye coordinates
- Projection transformation
  - Project 3D model to 2D plane
  - Perspective or orthographic projections
  - Specify viewing/clipping volume
- Viewport transformation
  - Scale "near" plane projection to window



Translation



Rotation



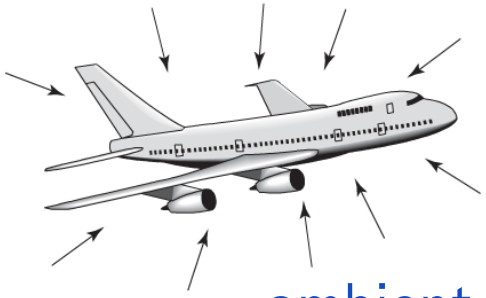
Scaling

3D Shapes with Material and Color

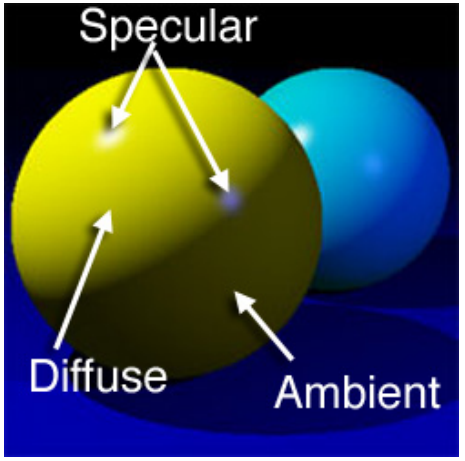
- Define interaction of light with 3D geometry
  - Specifies reflective properties of object surface
- Phong shaded material
  - Reflects ambient, diffuse, and specular light
  - Optionally, emits light
- Example
  - sphere.setMaterial(new PhongMaterial(Color.BLUE));
  - box.setMaterial(new PhongMaterial(Color.RED));
  - cylinder.setMaterial(new PhongMaterial(Color.GREEN));
  - Default lighting

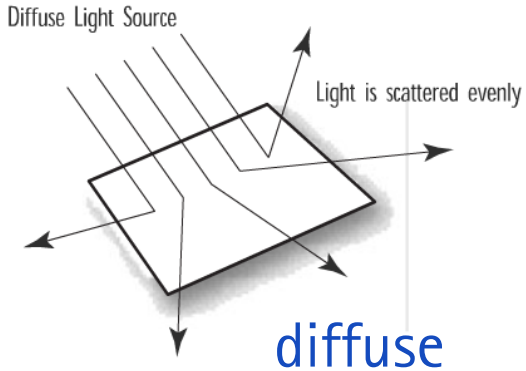
Light Components

- Ambient light
  - Light coming from any direction
- Diffuse light
  - Directional diffuse light source
  - Intensity proportional to angle
  - Scattered from surface
- Specular light
  - Directional light source
  - Uniformly reflected light
  - "Shininess" parameter:
    size of specular exponent
- Emission light
  - Light emitted from the material

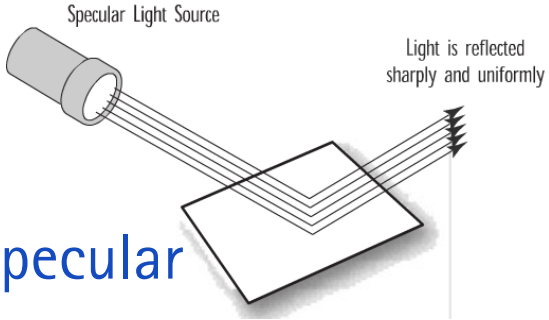


ambient





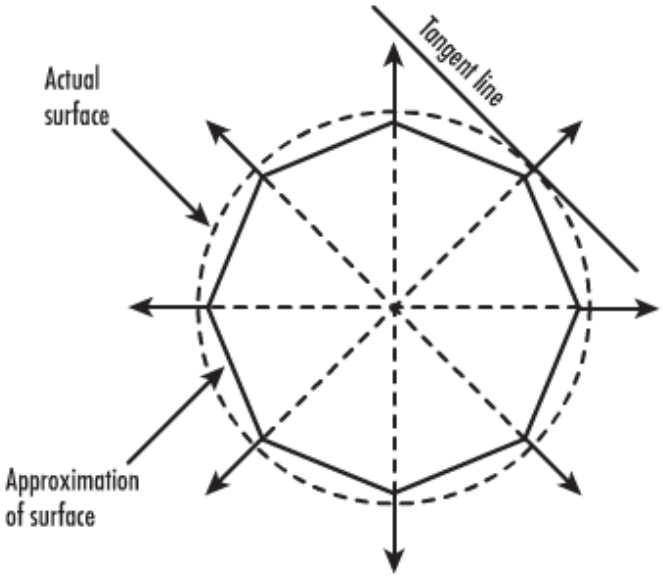
diffuse

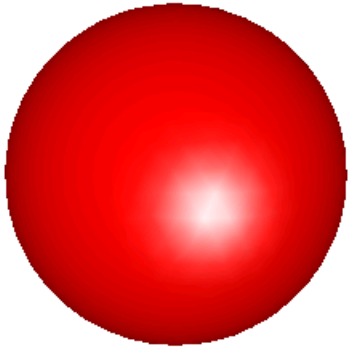


specular

Surface Normal Vectors

- A normal vector for each vertex
- A vertex may belong to multiple triangles
  - No unique surface normal for the vertex
- Interpolation across adjacent surfaces



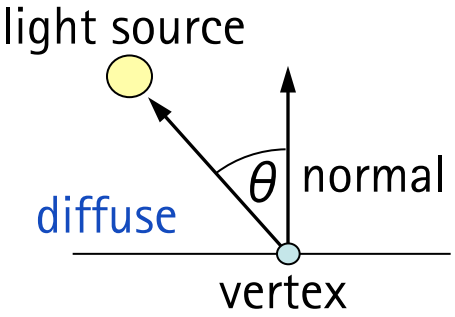


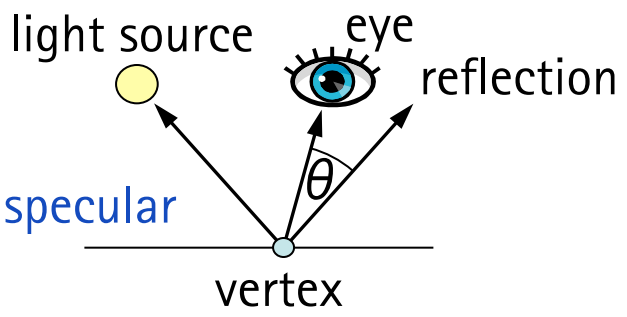
Smooth sphere (many vertices)

Rough sphere (few vertices)

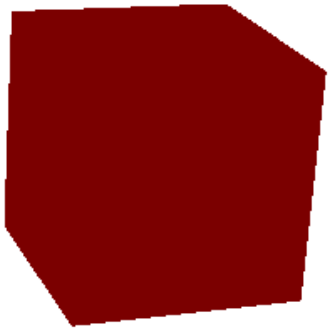
Material Reflectance Properties

- Material defines reflectance properties for the various light component intensities
  - Specify how material reflects ambient, diffuse, specular light
- Effect of light intensity I and material property m
  - Ambient: $I_a * m_a$
    - Example: light = (.5, .5, .3), material = (.5, 1, .2) ➔ (.25, .5, .06)
  - Diffuse: $I_d * m_d$ * dot product of vertex normal with normalized vertex-to-light vector
  - Specular: $I_s * m_s$ * dot product of normalized vertex-to-eye and vertex-to-reflection vectors, raised to the power of the material shininess
  - Attenuated by distance and other factors

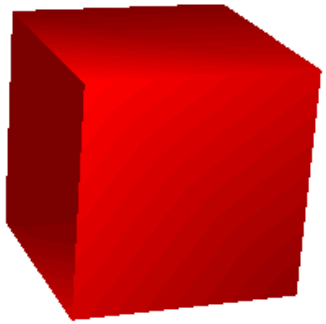




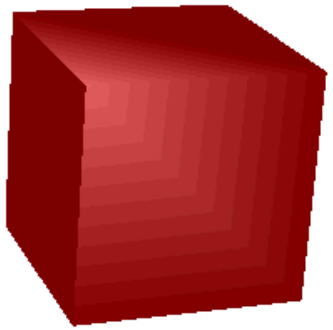
Effects of Lighting Components on Colored Material



Ambient light only



Ambient and diffuse light



Ambient diffuse, and specular light

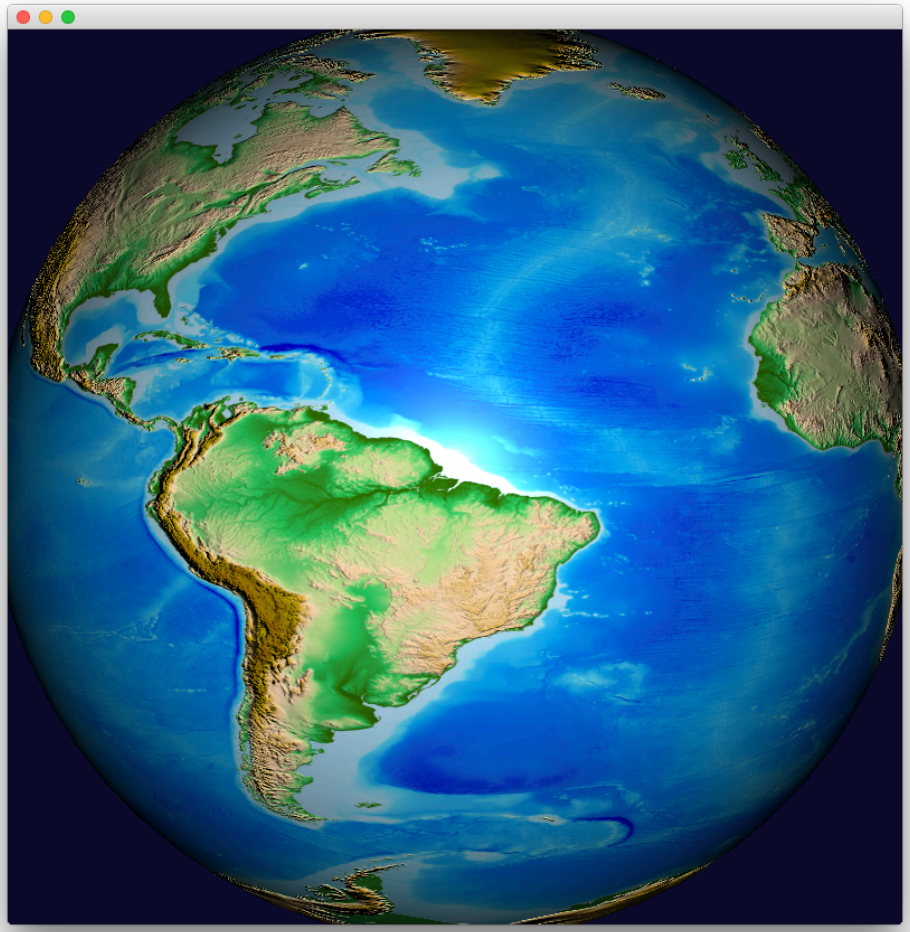
Placing Texture Images on 3D Geometry

- Images encode colors and surface normal vectors

- Example

```
PhongMaterial m = new PhongMaterial();
m.setBumpMap(new Image(path + "earth_normal.jpg"));
m.setDiffuseMap(new Image(path + "earth_diffuse.jpg"));
m.setSpecularMap(new Image(path + "earth_specular.jpg"));
s.setMaterial(m);
```

Rotating Earth



http://www.wthr.us/2013/05/23/bored-then-create-a-planet/
http://stackoverflow.com/questions/19621423/javafx-materials-bump-and-spec-maps

SCENE NODE ANIMATIONS

Transitions

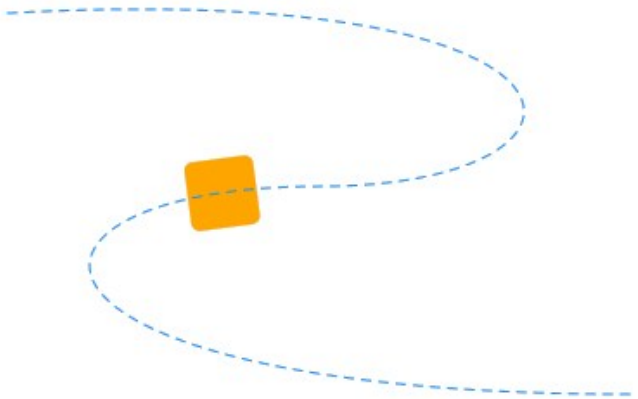
- Animations with an internal timeline
- Composable: Parallel or sequential execution

- Fade transition: Change node's opacity over time

```
FadeTransition ft = new FadeTransition(Duration.millis(1000), rect);
ft.setFromValue(1.0);
ft.setToValue(0.1);
ft.setCycleCount(Timeline.INDEFINITE);
ft.setAutoReverse(true);
ft.play();
```

From 1.0 (opaque) to 0.1 (almost transparent) in 1s, reverse (from 0.1 to 1.0 in 1s), repeat

Path Transitions



- Move along path

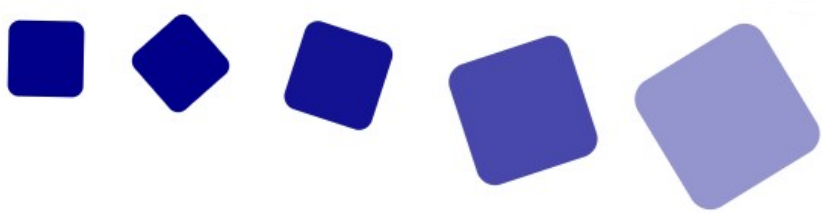
```
Path path = new Path();
path.getElements().add(new MoveTo(20, 20));
path.getElements().add(new CubicCurveTo(380, 0, 380, 120, 200, 120));
path.getElements().add(new CubicCurveTo(0, 120, 0, 240, 380, 240));

PathTransition pt = new PathTransition(3000, path, rect); // 3000 ms
pt.setOrientation(PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);
pt.setCycleCount(Timeline.INDEFINITE);
pt.setAutoReverse(true);
pt.play();
```

http://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/basics.htm

Parallel Transition

- Simultaneous execution

```
ParallelTransition pt = new ParallelTransition();
pt.getChildren().addAll(
        fadeTransition,
        translateTransition,
        rotateTransition,
        scaleTransition);
pt.setCycleCount(Timeline.INDEFINITE);
pt.play();
```



http://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/basics.htm

Sequential Transition

- Simultaneous execution

```
SequentialTransition st = new SequentialTransition();
st.getChildren().addAll(
        fadeTransition,
        translateTransition,
        rotateTransition,
        scaleTransition);
st.setCycleCount(Timeline.INDEFINITE);
st.play();
```

http://docs.oracle.com/javase/8/javafx/visual-effects-tutorial/basics.htm

Timeline Animation

- Animate arbitrary property values over time
- Key frame animation
  - Important (key) states (frames) for different times are defined
  - Intermediate values are automatically interpolated over time
- Example

```
Timeline timeline = new Timeline();
timeline.setCycleCount(Timeline.INDEFINITE);
timeline.setAutoReverse(true);
KeyValue kv = new KeyValue(rect.translateXProperty(), 300);
KeyFrame kf = new KeyFrame(Duration.millis(500), kv);
timeline.getKeyFrames().add(kf);
timeline.play();
```

Property: translateX
Value: 300
Duration: 500 ms

(initial value was 0)

Timeline Events

- Events may be triggered during timeline play
- On-finished event for timeline:

```
timeline.setOnFinished((ActionEvent e) -> {
        // timeline finished
});
```

- Timelines may be paused, played from a certain time, etc.

- On-finished event for key frame:

```
KeyFrame kf = new KeyFrame(Duration.millis(500), (ActionEvent e) -> {
        // key frame finished
}, kv);
```

Interpolators

- Defines function of time to property value
  - Linear, accelerated, decelerated, both, etc.
- Built-in behaviors
  - LINEAR
  - EASE_IN: acceleration at start
  - EASE_OUT: deceleration at end
  - EASE_BOTH
- May define arbitrary interpolators
  - i.e. a function that maps time [0,1] to value [0,1]