Kwin Effect Changes

Sept 2024, Akademy

David Edmundson

davidedmundson@kde.org
@d_ed

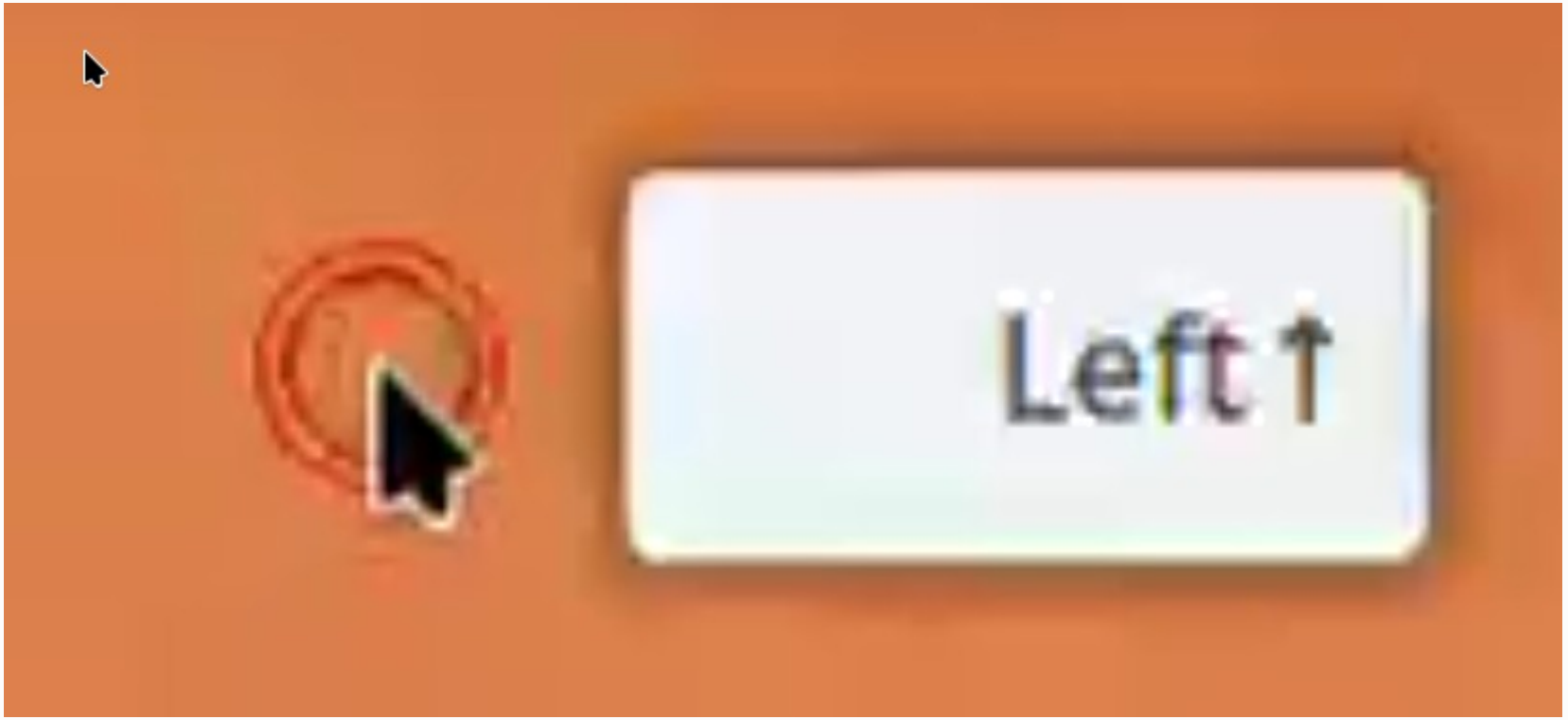
Kwin’s Old State – Massively Oversimplified

- Main code does drawing of everything using OpenGL
- Effects throw random OpenGL calls at the screen throughout

Example – MouseClick effect

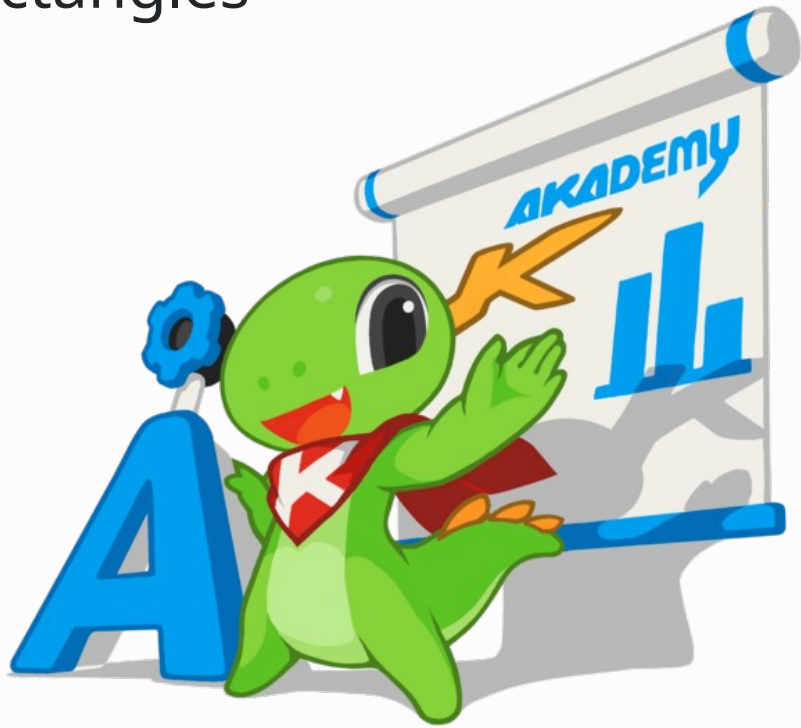
```
void MouseClickEffect::drawCircleGl(const RenderViewport &viewport, const QColor &color, float cx, float cy, float
r)
{
    static const int num_segments = 80;
    static const float theta = 2 * 3.1415926 / float(num_segments);
    static const float c = cosf(theta); // precalculate the sine and cosine
    static const float s = sinf(theta);
    const float scale = viewport.scale();
    float t;

    float x = r; // we start at angle = 0
    float y = 0;

    GLVertexBuffer *vbo = GLVertexBuffer::streamingBuffer();
    vbo->reset();
    QList<QVector2D> verts;
    verts.reserve(num_segments * 2);

    for (int ii = 0; ii < num_segments; ++ii) {
        verts.push_back(QVector2D((x + cx) * scale, (y + cy) * scale)); // output vertex
        // apply the rotation matrix
        t = x;
        x = c * x - s * y;
        y = s * t + c * y;
    }
    vbo->setVertices(verts);
    ShaderManager::instance()->getBoundShader()->setUniform(GLShader::ColorUniform::Color, color);
    vbo->render(GL_LINE_LOOP);
}
```

Example – End Result



What the core devs want

- Really low level API
- Light-weight scenegraph of simple rectangles
- Path to Vulkan



What the effect developers want

- Really high level API
- Complex shapes and imports
- Not to write separate Vulkan code

Solution!

The solution is to have two solutions!

Mechanism

- Effects use QtQuick to render into a texture
- Easy to use items to represent Windows and Desktops

- Core render code renders these textures with zero-copy

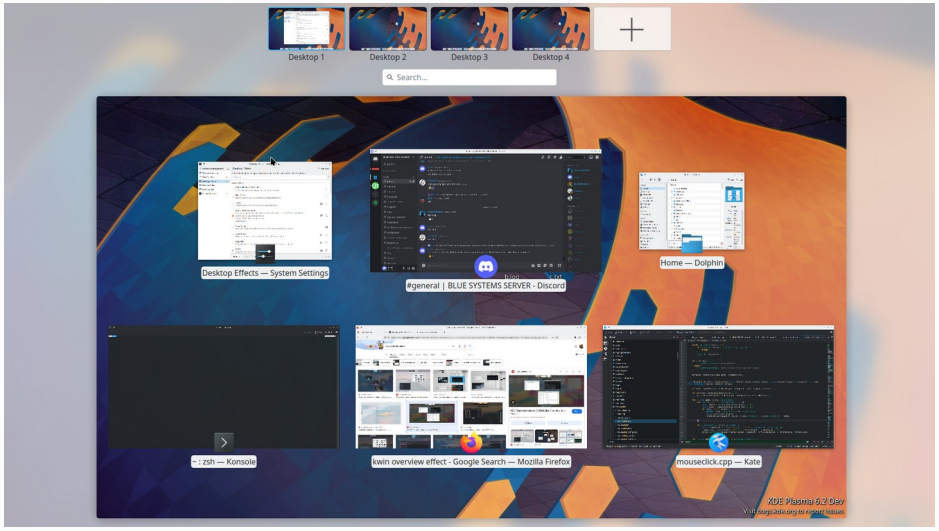
Benefits

Benefits – Easy to Code

- Designers can contribute to KWin
- Features like anti-aliasing for free
- Readable code

```
Rectangle {
    width: mySize
    radius: width/2
}
```

Benefits - Interactivity

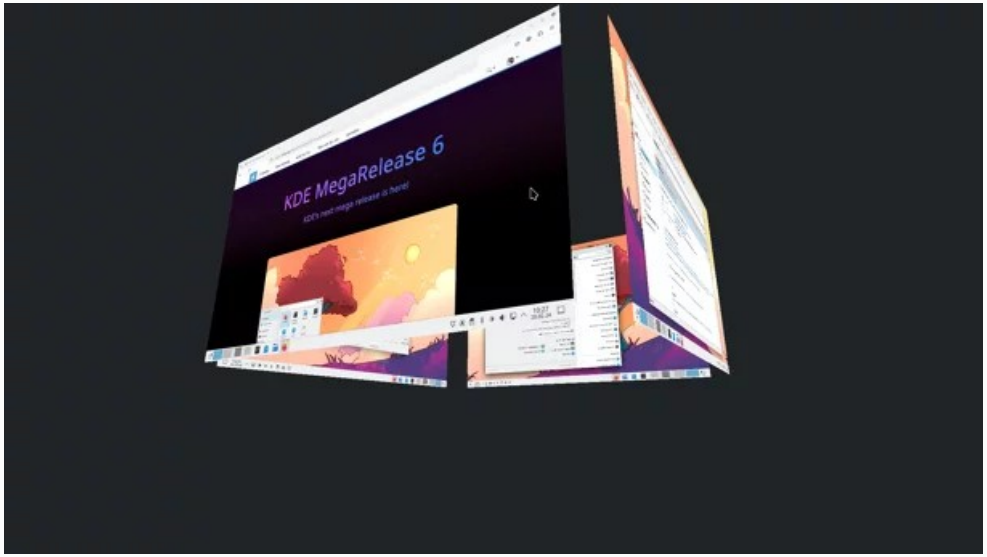


- Adding interactivity is easy
- We can add drag and drop
- Krunner integration was easy

Benefits - 3D

- QtQuick 3D is amazing!
- Relatively easy to use

.



Example – Next Level

```
View3D {
    id: view
    anchors.fill: parent
    renderMode: View3D.Underlay

    PerspectiveCamera {
        position: Qt.vector3d(0, 25, 50)
        eulerRotation.x: -30
    }

    DirectionalLight {
        eulerRotation.x: -30
    }

    // Generated from running `balsam hand.stl`
    Hand {
        id: hand
        z: -25

        Vector3dAnimation on eulerRotation {
            loops: Animation.Infinite
            duration: 5000
            from: Qt.vector3d(45, 90, 90)
            to: Qt.vector3d(45, 90, -90)
        }
    }
}
```

And Particles!

And 3D particles!

What will you write?