Understanding DPI Scaling in Qt C++

The Core Problem

The key thing to understand is that **screen resolution alone doesn't tell you about DPI scaling**. What matters is the **physical size of the screen** combined with its resolution.

Example: Three Different Monitors

Let's say you have three monitors:

1. **24" monitor at 1920×1080** → ~92 DPI (normal density)
2. **13" laptop at 1920×1080** → ~170 DPI (high density)
3. **27" monitor at 3840×2160** → ~163 DPI (4K/high density)

Even though monitors 1 and 2 have the same resolution, the laptop screen packs way more pixels into a smaller physical space. If you draw a button at "100 pixels wide" on both:
- On the 24" monitor: looks fine, readable
- On the 13" laptop: tiny, illegible (same 100 pixels in half the physical space)

This is what DPI (dots per inch) measures — how densely packed the pixels are.

What Likely Happened With Your 1366×768 App

When you developed on a 1366×768 screen, you probably:
- Hard-coded some widget sizes in pixels
- Positioned things with fixed coordinates
- Used fixed font sizes or didn't test scaling

Now when you run it on different screens:
- **1920×1080 screen**: Might work fine IF it has similar DPI (~96), but widgets may look small if the screen is physically larger
- **3840×2160 screen**: Almost certainly looks tiny because 4K screens typically have 2x the DPI (200% scaling in Windows)

The Solution: Make Your App DPI-Independent

Here's how to port your app properly:

1. Enable High DPI Scaling (Critical First Step)

In your `main.cpp`, **before** creating `QApplication`:

```
#include &lt;QApplication&gt;
#include &lt;QGuiApplication&gt;

int main(int argc, char *argv[]) {
    // Qt 5.6+ - MUST be before QApplication
    QApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    QApplication::setAttribute(Qt::AA_UseHighDpiPixmaps);

    // Optional: allow fractional scaling for more accurate results
    QGuiApplication::setHighDpiScaleFactorRoundingPolicy(
        Qt::HighDpiScaleFactorRoundingPolicy::PassThrough
    );

    QApplication app(argc, argv);

    // Your app code...

    return app.exec();
}
```

In Qt 6, this is automatic — you don't need these lines.

This tells Qt to automatically scale everything based on the system DPI settings. Windows will report "this is a 200% DPI screen" and Qt will scale accordingly.

---

2. Replace All Hard-Coded Pixel Sizes

Search your codebase for these anti-patterns:

Bad - Hard-coded positions and sizes:

```
button-&gt;setGeometry(50, 100, 120, 30);  // Fixed position and size
widget-&gt;setFixedSize(300, 200);
widget-&gt;move(100, 50);
layout-&gt;setContentsMargins(10, 10, 10, 10);  // Fixed margins
```

Good - Use layouts and let Qt calculate sizes:

```
// Use layouts - they handle sizing automatically
QVBoxLayout *layout = new QVBoxLayout;
layout-&gt;addWidget(button);

// If you MUST set sizes, use font-relative sizing
QFontMetrics fm(button-&gt;font());
int em = fm.height();  // One line of text height - DPI aware!

button-&gt;setMinimumWidth(em * 8);    // 8 character widths
button-&gt;setMinimumHeight(em * 2);   // 2 lines tall
layout-&gt;setContentsMargins(em/2, em/2, em/2, em/2);  // Relative margins
```

---

3. Fix Your Layout Structure

If your app was designed with fixed positions, you need to convert it to use Qt's layout system:

Original (broken) approach:

```
MainWindow::MainWindow() {
    QWidget *central = new QWidget(this);
    setCentralWidget(central);

    // Fixed positions - breaks on different screens
    QPushButton *btn1 = new QPushButton("Button 1", central);
    btn1-&gt;setGeometry(10, 10, 100, 30);

    QLabel *label = new QLabel("Label", central);
    label-&gt;setGeometry(10, 50, 200, 20);

    QLineEdit *input = new QLineEdit(central);
    input-&gt;setGeometry(10, 80, 200, 25);
}
```

Converted to layouts (works on all screens):

```
MainWindow::MainWindow() {
    QWidget *central = new QWidget(this);
    setCentralWidget(central);

    QVBoxLayout *layout = new QVBoxLayout(central);

    // Widgets automatically sized by layout
    QPushButton *btn1 = new QPushButton("Button 1");
    layout-&gt;addWidget(btn1);

    QLabel *label = new QLabel("Label");
    layout-&gt;addWidget(label);

    QLineEdit *input = new QLineEdit;
    layout-&gt;addWidget(input);

    // Set reasonable margins based on font size
    QFontMetrics fm(QApplication::font());
    int em = fm.height();
    layout-&gt;setContentsMargins(em/2, em/2, em/2, em/2);
    layout-&gt;setSpacing(em/4);
}
```

---

4. Fix Stylesheets with Hard-Coded Pixels

If you use Qt Style Sheets with pixel values, they won't scale:

Bad - Fixed pixel values:

```
button-&gt;setStyleSheet(
    "QPushButton {"
    "    padding: 8px 16px;"
    "    min-height: 32px;"
    "    font-size: 12px;"
    "}"
);
```

Good - Calculate from font metrics:

```
QFontMetrics fm(button-&gt;font());
int em = fm.height();

QString style = QString(
```

```
    "QPushButton {"
    "    padding: %1px %2px;"
    "    min-height: %3px;"
    "}"
).arg(em / 4)      // padding vertical
 .arg(em / 2)      // padding horizontal
 .arg(em * 2);     // min height

button-&gt;setStyleSheet(style);
```

Don't set `font-size` in stylesheets — let the system font size (which is already DPI-aware) do the work.

---

5. Icon Sizes

If you have icons, make sure they scale too:

```
// Bad - fixed icon size
button-&gt;setIconSize(QSize(16, 16));

// Good - scale with font
QFontMetrics fm(button-&gt;font());
int em = fm.height();
button-&gt;setIconSize(QSize(em, em));

// Even better - provide multiple icon resolutions
// Qt automatically picks the right one for the DPI
QIcon icon;
icon.addFile(":/icons/myicon-16.png");   // Normal DPI
icon.addFile(":/icons/myicon-32.png");   // 2x DPI
icon.addFile(":/icons/myicon-48.png");   // 3x DPI
button-&gt;setIcon(icon);
```

---

6. Images and Pixmaps

For images displayed in your app:

```
// Load image with DPI awareness
QPixmap pixmap(":/images/logo.png");
pixmap.setDevicePixelRatio(qApp-&gt;devicePixelRatio());

// Or load the @2x version automatically on high-DPI
// Qt looks for logo@2x.png on 2x DPI screens
QPixmap pixmap(":/images/logo.png");  // Automatically loads logo@2x.png if available
```

Testing Your Changes

Test with DPI Scale Factors

You can force different scale factors without changing monitors:

Windows:

```
set QT_SCALE_FACTOR=1.0
myapp.exe

set QT_SCALE_FACTOR=1.5
myapp.exe

set QT_SCALE_FACTOR=2.0
myapp.exe
```

Linux/macOS:

```
QT_SCALE_FACTOR=1.0 ./myapp
QT_SCALE_FACTOR=1.5 ./myapp
QT_SCALE_FACTOR=2.0 ./myapp
```

Run your app with each factor and verify:
- Text doesn't overflow buttons
- Widgets don't overlap
- Spacing looks proportional
- Nothing is cut off

Real World Example: Before and After

Before (Fixed pixels - broken):

```
class MainWindow : public QMainWindow {
public:
    MainWindow() {
        setFixedSize(800, 600);  // Fixed window size

        QWidget *central = new QWidget(this);
        setCentralWidget(central);

        // All fixed positions
        title = new QLabel("My Application", central);
        title-&gt;setGeometry(20, 20, 760, 40);
        title-&gt;setStyleSheet("font-size: 24px;");  // Fixed font

        nameLabel = new QLabel("Name:", central);
        nameLabel-&gt;setGeometry(20, 80, 100, 25);

        nameInput = new QLineEdit(central);
        nameInput-&gt;setGeometry(130, 80, 200, 25);

        submitBtn = new QPushButton("Submit", central);
        submitBtn-&gt;setGeometry(20, 120, 100, 30);
    }

private:
    QLabel *title;
    QLabel *nameLabel;
    QLineEdit *nameInput;
    QPushButton *submitBtn;
};
```

After (Layout-based - works on all screens):

```
class MainWindow : public QMainWindow {
public:
    MainWindow() {
        // Let the window size itself based on content
        resize(800, 600);  // Initial size hint, not fixed

        QWidget *central = new QWidget(this);
        setCentralWidget(central);

        // Use layouts
        QVBoxLayout *mainLayout = new QVBoxLayout(central);

        // Get base sizing unit
        QFontMetrics fm(QApplication::font());
        int em = fm.height();

        // Title with font-relative size
        title = new QLabel("My Application");
        QFont titleFont = title-&gt;font();
        titleFont.setPointSize(titleFont.pointSize() * 1.5);  // Relative size
        title-&gt;setFont(titleFont);
        title-&gt;setAlignment(Qt::AlignCenter);
        mainLayout-&gt;addWidget(title);

        mainLayout-&gt;addSpacing(em);  // Relative spacing

        // Form layout for input
        QFormLayout *formLayout = new QFormLayout;
        nameInput = new QLineEdit;
        formLayout-&gt;addRow("Name:", nameInput);
        mainLayout-&gt;addLayout(formLayout);

        mainLayout-&gt;addSpacing(em / 2);

        // Button row
        QHBoxLayout *btnLayout = new QHBoxLayout;
        btnLayout-&gt;addStretch();
        submitBtn = new QPushButton("Submit");
        submitBtn-&gt;setMinimumHeight(em * 2);  // Relative size
        btnLayout-&gt;addWidget(submitBtn);
        mainLayout-&gt;addLayout(btnLayout);

        mainLayout-&gt;addStretch();

        // Relative margins
        mainLayout-&gt;setContentsMargins(em, em, em, em);
        mainLayout-&gt;setSpacing(em / 2);
    }

private:
    QLabel *title;
    QLineEdit *nameInput;
    QPushButton *submitBtn;
};
```

Common Pitfalls When Porting

1. **Forgetting to enable high DPI in main()** - Without this, nothing else matters

2. **Using Designer with pixel coordinates** - If you use Qt Designer, make sure you're using layouts, not absolute positioning
3. **Mixing fixed sizes with layouts** - If one widget has `setFixedSize()`, it breaks the layout's ability to adapt
4. **Testing only on your development screen** - Always test with `QT_SCALE_FACTOR`
5. **Assuming resolution = DPI** - A 1920×1080 laptop has very different DPI than a 1920×1080 desktop monitor

---

Quick Checklist for Your Port

Use this checklist when porting your 1366×768 app to work on all screen resolutions:

- [ ] Add `QApplication::setAttribute(Qt::AA_EnableHighDpiScaling)` to main()
- [ ] Search code for `setGeometry()` - replace with layouts
- [ ] Search for `setFixedSize()` - replace with `setMinimumSize()` using em units
- [ ] Search for `.move()` - replace with layout positioning
- [ ] Search stylesheets for `px` values - replace with runtime-calculated values
- [ ] Search for `setIconSize()` - make relative to font metrics
- [ ] Convert any absolute positions to layouts (QVBoxLayout, QHBoxLayout, QGridLayout)
- [ ] Test with `QT_SCALE_FACTOR=1.0`, `1.5`, and `2.0`
- [ ] Test on actual 1920×1080 and 4K screens if possible

Specific Case: Porting from 1366×768 to Multiple Resolutions

Your Scenario

You have an app designed for **1366×768** and need it to work on:
- **1920×1080** screens
- **3840×2160** (4K) screens

What You Need to Do

1. **Enable DPI scaling in main()**
- This is the single most important step
- Add the attributes before creating QApplication

2. **Audit your code for fixed sizes**
- Search for: `setGeometry`, `setFixedSize`, `setFixedWidth`, `setFixedHeight`, `move`
- Replace all with layout-based positioning

3. **Convert to layout system**
- If you used absolute positioning, switch to QVBoxLayout, QHBoxLayout, QGridLayout, QFormLayout
- Use `addWidget()` instead of setting x/y coordinates

4. **Make sizes relative**
- Calculate an `em` unit from font metrics
- Use multiples of `em` for all spacing, margins, minimum sizes

5. **Test thoroughly**
- Use `QT_SCALE_FACTOR` environment variable to simulate different DPIs

- Test at 1.0x (96 DPI), 1.5x (144 DPI), and 2.0x (192 DPI)

Expected Results

Once you complete the port:
- **On 1920×1080 at 96 DPI**: App looks similar to original, maybe slightly larger due to more space
- **On 1920×1080 at 144 DPI** (high-DPI laptop): App scales up 1.5x, remains readable
- **On 3840×2160 at 192 DPI**: App scales up 2x, text and buttons remain perfectly readable
- **Future-proof**: Will work on any screen size and DPI combination

---

Advanced: Per-Monitor DPI Awareness

If your app moves between monitors with different DPIs (e.g., laptop + external 4K monitor), you may want to handle DPI changes dynamically:

```
class MainWindow : public QMainWindow {
    Q_OBJECT
public:
    MainWindow() {
        applyDpiAwareStyles();
    }

protected:
    void changeEvent(QEvent *event) override {
        if (event-&gt;type() == QEvent::ScreenChangeEvent) {
            // Window moved to a different screen with different DPI
            applyDpiAwareStyles();
        }
        QMainWindow::changeEvent(event);
    }

private:
    void applyDpiAwareStyles() {
        QFontMetrics fm(QApplication::font());
        int em = fm.height();

        QString style = QString(
            "QPushButton { padding: %1px %2px; min-height: %3px; }"
            "QLineEdit { padding: %4px %2px; }"
        ).arg(em / 4).arg(em / 2).arg(em * 2).arg(em / 6);

        setStyleSheet(style);
    }
};
```

Summary

The key insight is that **resolution is not the same as DPI**. A 1920×1080 laptop screen has very different physical dimensions than a 1920×1080 desktop monitor, resulting in different DPIs.

To make your Qt app work across all screens:

1. **Enable high DPI scaling** - One line in main(), critical
2. **Use layouts, not fixed positions** - Let Qt calculate sizes
3. **Make all sizes relative to font metrics** - Use `em` units
4. **Test with different scale factors** - Use `QT_SCALE_FACTOR`

Once you do this work, your app will automatically adapt to **any** screen resolution and DPI — not just the three you mentioned, but future 8K screens, ultrawide monitors, different laptops, tablets, and any other display technology that comes along.