

Floating-Point Formats for Textures, Framebuffers, and Renderbuffers

This appendix will describe the floating-point formats used for pixel storage in framebuffers and renderbuffers, and texel storage in textures. It has the following major sections:

- “Floating-Point Formats Used in OpenGL”

Floating-Point Formats Used in OpenGL

In addition to the normal 32-bit single-precision floating-point values you are used to using when you declare a GLfloat in your application, OpenGL supports reduced-precision floating-point representations for storing data more compactly than its 32-bit representation. In many instances, your floating-point data may not require the entire dynamic range of a 32-bit float, and storing or processing data in a reduced-precision format may save memory, and increase data transfer rates.

OpenGL supports three reduce-precision floating-point formats: 16-bit (signed) floating-point values, and 10- and 11-bit unsigned floating-point values. Table J-1 describes the bit-layout of each representation, and the associated pixel formats

Floating-point Type	Associated Pixel Formats	Sign bit	Number of Exponents Bits	Number of Mantissa Bits
16-bit	GL_RGB16F, GL_RGBA16F	1	5	10
11-bit	GL_R11F_G11F_B10F (red and green components)	0	5	6
10-bit	GL_R11F_G11F_B10F (blue component)	0	5	5

Table J-1 Reduced-Precision Floating-Point Formats

16-bit Floating-Point Values

For signed 16-bit floating-point values, the minimum and maximum values that can be represented are (about) 6.103×10^{-5} , and 65504.0, respectively.

The following routine, **F32toF16()**, will convert a single, full-precision 32-bit floating-point value to a 16-bit reduced-precision form (stored as an unsigned-short integer)

```
#define F16_EXPONENT_BITS    0x1F
#define F16_EXPONENT_SHIFT  10
#define F16_EXPONENT_BIAS   15
#define F16_MANTISSA_BITS    0x3ff
#define F16_MANTISSA_SHIFT   (23 - F16_EXPONENT_SHIFT)
#define F16_MAX_EXPONENT    \
    (F16_EXPONENT_BITS << F16_EXPONENT_SHIFT)
```

```

GLushort
F32toF16(GLfloat val)
{
    GLuint    f32 = (*(GLuint *) &val);
    GLushort  f16 = 0;

    /* Decode IEEE 754 little-endian 32-bit floating-point value
    */
    int sign    = (f32 >> 16) & 0x8000;
    /* Map exponent to the range [-127,128] */
    int exponent = ((f32 >> 23) & 0xff) - 127;
    int mantissa = f32 & 0x007fffff;

    if (exponent == 128) { /* Infinity or NaN */
        f16 = sign | F16_MAX_EXPONENT;
        if (mantissa) f16 |= (mantissa & F16_MANTISSA_BITS);
    }
    else if (exponent > 15) { /* Overflow - flush to Infinity */
        f16 = sign | F16_MAX_EXPONENT;
    }
    else if (exponent > -15) { /* Representable value */
        exponent += F16_EXPONENT_BIAS;
        mantissa >>= F16_MANTISSA_SHIFT;
        f16 = sign | exponent << F16_EXPONENT_SHIFT | mantissa;
    }
    else {
        f16 = sign;
    }

    return f16;
}

```

Likewise, **F16toF32()** converts from the reduced-precision floating-point form into a normal 32-bit floating-point value.

```

#define F32_INFINITY 0x7f800000

GLfloat
F16toF32(GLushort val)
{
    union {
        GLfloat f;
        GLuint ui;
    } f32;

    int sign    = (val & 0x8000) << 15;
    int exponent = (val & 0x7c00) >> 10;
    int mantissa = (val & 0x03ff);

```

```

f32.f = 0.0;

if (exponent == 0) {
    if (mantissa != 0) {
        const GLfloat scale = 1.0 / (1 << 24);
        f32.f = scale * mantissa;
    }
}
else if (exponent == 31) {
    f32.ui = sign | F32_INFINITY | mantissa;
}
else {
    GLfloat scale, decimal;
    exponent -= 15;
    if (exponent < 0) {
        scale = 1.0 / (1 << -exponent);
    }
    else {
        scale = 1 << exponent;
    }
    decimal = 1.0 + (float) mantissa / (1 << 10);
    f32.f = scale * decimal;
}

if (sign) f32.f = -f32.f;

return f32.f;
}

```

10- and 11-bit Unsigned Floating-Point Values

For normalized color values in the range [0, 1], unsigned 10- and 11-bit floating-point formats may provide a more compact format with better dynamic range than either floating-point values, or OpenGL's unsigned integer pixel formats. The maximum representable values are 65204 and 64512, respectively.

Routines for converting floating-point values into 10-bit unsigned floating-point values, and vice versa are shown below.

```

#define UF11_EXPONENT_BIAS    15
#define UF11_EXPONENT_BITS   0x1F
#define UF11_EXPONENT_SHIFT   6
#define UF11_MANTISSA_BITS   0x3F
#define UF11_MANTISSA_SHIFT  (23 - UF11_EXPONENT_SHIFT)

```

```

#define UF11_MAX_EXPONENT \
    (UF11_EXPONENT_BITS << UF11_EXPONENT_SHIFT)

GLushort
F32toUF11(GLfloat val)
{
    GLuint    f32 = *(GLuint *) &val);
    GLushort  uf11 = 0;

    /* Decode little-endian 32-bit floating-point value */
    int sign    = (f32 >> 16) & 0x8000;
    /* Map exponent to the range [-127,128] */
    int exponent = ((f32 >> 23) & 0xff) - 127;
    int mantissa = f32 & 0x007fffff;

    if (sign) return 0;

    if (exponent == 128) { /* Infinity or NaN */
        uf11 = UF11_MAX_EXPONENT;
        if (mantissa) uf11 |= (mantissa & UF11_MANTISSA_BITS);
    }
    else if (exponent > 15) { /* Overflow - flush to Infinity */
        uf11 = UF11_MAX_EXPONENT;
    }
    else if (exponent > -15) { /* Representable value */
        exponent += UF11_EXPONENT_BIAS;
        mantissa >>= UF11_MANTISSA_SHIFT;
        uf11 = exponent << UF11_EXPONENT_SHIFT | mantissa;
    }

    return uf11;
}

#define F32_INFINITY 0x7f800000

GLfloat
UF11toF32(GLushort val)
{
    union {
        GLfloat f;
        GLuint  ui;
    } f32;

    int exponent = (val & 0x07c0) >> UF11_EXPONENT_SHIFT;
    int mantissa = (val & 0x003f);

```

```

f32.f = 0.0;

if (exponent == 0) {
    if (mantissa != 0) {
        const GLfloat scale = 1.0 / (1 << 20);
        f32.f = scale * mantissa;
    }
}
else if (exponent == 31) {
    f32.ui = F32_INFINITY | mantissa;
}
else {
    GLfloat scale, decimal;
    exponent -= 15;
    if (exponent < 0) {
        scale = 1.0 / (1 << -exponent);
    }
    else {
        scale = 1 << exponent;
    }
    decimal = 1.0 + (float) mantissa / 64;
    f32.f = scale * decimal;
}

return f32.f;
}

```

For completeness, we present similar routines for converting 10-bit unsigned floating-point values.

```

#define UF10_EXPONENT_BIAS    15
#define UF10_EXPONENT_BITS   0x1F
#define UF10_EXPONENT_SHIFT   5
#define UF10_MANTISSA_BITS   0x3F
#define UF10_MANTISSA_SHIFT  (23 - UF10_EXPONENT_SHIFT)
#define UF10_MAX_EXPONENT    \
    (UF10_EXPONENT_BITS << UF10_EXPONENT_SHIFT)

GLushort
F32toUF10(GLfloat val)
{
    GLuint    f32 = (*(GLuint *) &val);
    GLushort  uf10 = 0;

    /* Decode little-endian 32-bit floating-point value */
    int sign    = (f32 >> 16) & 0x8000;

```

```

/* Map exponent to the range [-127,128] */
int exponent = ((f32 >> 23) & 0xff) - 127;
int mantissa = f32 & 0x007fffff;

if (sign) return 0;

if (exponent == 128) { /* Infinity or NaN */
    uf10 = UF10_MAX_EXPONENT;
    if (mantissa) uf10 |= (mantissa & UF10_MANTISSA_BITS);
}
else if (exponent > 15) { /* Overflow - flush to Infinity */
    uf10 = UF10_MAX_EXPONENT;
}
else if (exponent > -15) { /* Representable value */
    exponent += UF10_EXPONENT_BIAS;
    mantissa >>= UF10_MANTISSA_SHIFT;
    uf10 = exponent << UF10_EXPONENT_SHIFT | mantissa;
}

return uf10;
}

#define F32_INFINITY 0x7f800000

GLfloat
UF10toF32(GLushort val)
{
    union {
        GLfloat f;
        GLuint ui;
    } f32;

    int exponent = (val & 0x07c0) >> UF10_EXPONENT_SHIFT;
    int mantissa = (val & 0x003f);

    f32.f = 0.0;

    if (exponent == 0) {
        if (mantissa != 0) {
            const GLfloat scale = 1.0 / (1 << 20);
            f32.f = scale * mantissa;
        }
    }
    else if (exponent == 31) {
        f32.ui = F32_INFINITY | mantissa;
    }
}

```

```
    else {
        GLfloat scale, decimal;
        exponent -= 15;
        if (exponent < 0) {
            scale = 1.0 / (1 << -exponent);
        }
        else {
            scale = 1 << exponent;
        }
        decimal = 1.0 + (float) mantissa / 64;
        f32.f = scale * decimal;
    }

    return f32.f;
}
```