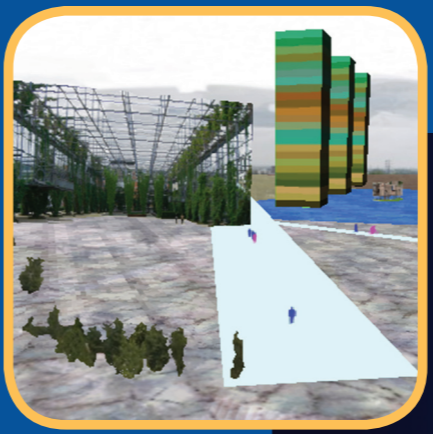


```

RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformPartialRevCubic( result, VM::toInt( state, 2 ) );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
fastFourierTransformVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->fastFourierTransform( result );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
maxAmplitudeVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->maxAmplitude( result, VM::toInt( state, 2 ), VM::toInt( state, 3 ), VM::toInt( state, 4 ) );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
translateToMatchVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<const FCurveUniformSamples> other( FCurveUniformSamplesVM::toConst( state, 2 ) );
float scale, offset;
int translate = curve->translateToMatch( other, scale, offset );
VM::push( state, translate );
VM::push( state, scale );
VM::push( state, offset );
return 3;
}
CLASS RevOrdering
class RevOrdering
public:
bool operator() ( float a, float b ) const
{
return ( a < b );
}
};
class HaarWaveletOrdering
public:
/*--- methods ---*/
HaarWaveletOrdering( const FCurveUniformSamples::Container& waveletCoefficients )
: _waveletCoefficients( waveletCoefficients )
{
if ( _waveletMultiplier.size() == waveletCoefficients.size() )
{
return;
}
_waveletMultiplier.resize( waveletCoefficients.size() );
uint increment;
for ( increment=1; increment < _waveletMultiplier.size()-1; increment<<=1 )
{
uint waveletIdx;
for ( waveletIdx=0; waveletIdx < _waveletMultiplier.size(); waveletIdx += increment )
{
if ( increment == 1 )

```

Graphics Interface 2009

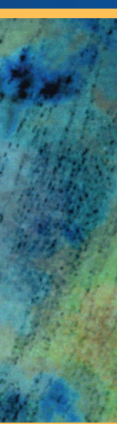
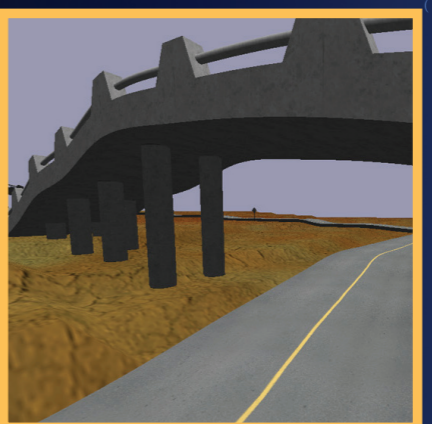
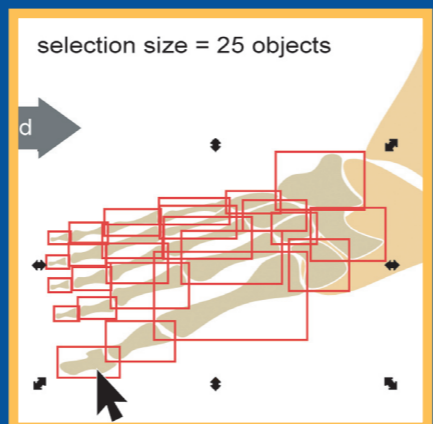
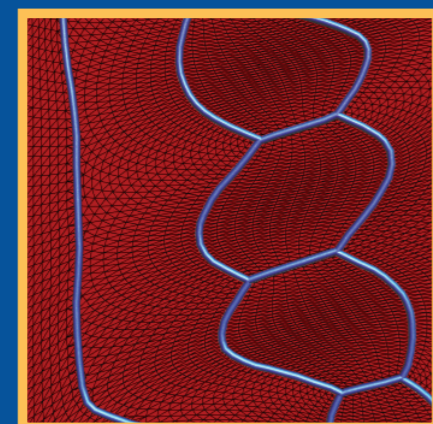


```

RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->resampleLinear( result, VM::toInt( state, 2 ) );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
waveletTransformFwdHaarVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformFwdHaar( result );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
waveletTransformRevHaarVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformRevHaar( result );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
waveletTransformPartialRevHaarVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformPartialRevHaar( result, VM::toInt( state, 2 ) );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
waveletTransformFwdLinearVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformFwdLinear( result );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
waveletTransformRevLinearVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformRevLinear( result );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
waveletTransformPartialRevLinearVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformPartialRevLinear( result, VM::toInt( state, 2 ) );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
waveletTransformFwdCubicVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformFwdCubic( result );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
waveletTransformRevCubicVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformRevCubic( result );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
int
waveletTransformPartialRevCubicVM
( VMState* state )
{
RCP<const FCurveUniformSamples> curve( FCurveUniformSamplesVM::toConst( state, 1 ) );
RCP<FCurveUniformSamples> result;
curve->waveletTransformPartialRevCubic( result, VM::toInt( state, 2 ) );
FCurveUniformSamplesVM::push( state, result );
return 1;
}
}

```

Proceedings
Graphics Interface 2009
25 - 27 May 2009
Kelowna, British Columbia, Canada
Canadian Human-Computer Communications Society/
Société Canadienne du Dialogue Humaine Machine
(CHCCS/SCDHM)



```

waveletMultiplier[waveletIdx] = 1;
else
_waveletMultiplier *= 2;
TRACE( "C << i << ", " << _waveletMultiplier[i] << " );
bool operator() ( uint a, uint b ) const
{
return ( a < b );
}
};

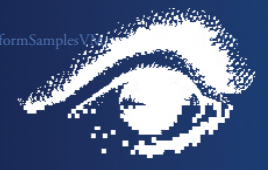
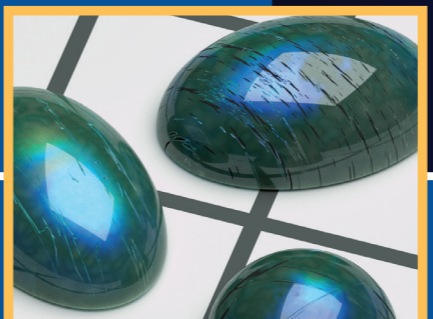
```



www.graphicsinterface.org
www.akpeters.com



A K PETERS



IEEE
vgtc

A K PETERS LTD.

ISSN 0713-5424
ISBN 978-1-56881-470-4