Qt 6.8.1 with OpenCV 4.10.0 Installation Manual

Table of Contents

1. System Requirements

Hardware Requirements

- NVIDIA GPU with CUDA support
- 32GB RAM recommended
- 50GB+ free disk space

Software Requirements

- Windows 10/11 (64-bit)
- Visual Studio 2022 (MSVC 17.0)
- CUDA Toolkit (version matching your nvidia-smi output)
- Qt 6.8.1
- OpenCV 4.10.0
- CMake 3.27 or later

2. Installation Steps

2.1 Install Visual Studio 2022

1. Download from Microsoft website
2. Select components:
    - Desktop development with C++
    - MSVC v143 build tools (x64/x86)
    - Windows 10/11 SDK
    - C++ CMake tools for Windows

2.2 CUDA and cuDNN Setup

2.2.1 Check CUDA Compatibility

1. Check your current CUDA setup:

```
nvidia-smi
```

Example output:

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 535.98    Driver Version: 535.98    CUDA Version: 12.2     |
|-------------------------------+----------------------+----------------------|
```

2. Install matching CUDA version from NVIDIA website

    - Go to: https://developer.nvidia.com/cuda-toolkit-archive
    - Select the version matching your nvidia-smi output
    - Download and install CUDA Toolkit

2.2.2 cuDNN Installation

1. Download matching cuDNN:

    - Go to: https://developer.nvidia.com/cudnn
    - Find cuDNN version compatible with your CUDA version
    - Download cuDNN (requires NVIDIA account)

2. Install cuDNN:

    - Extract the downloaded cuDNN archive
    - Copy files to CUDA installation directory:

      ```
      Copy <cuDNN>/bin/* to C:/Program Files/NVIDIA GPU Computing
      Toolkit/CUDA/v12.x/bin
      Copy <cuDNN>/include/* to C:/Program Files/NVIDIA GPU Computing
      Toolkit/CUDA/v12.x/include
      Copy <cuDNN>/lib/* to C:/Program Files/NVIDIA GPU Computing
      Toolkit/CUDA/v12.x/lib
      ```

    - Replace v12.x with your CUDA version

3. Verify Installation:

    - Check CUDA version: `nvcc --version`
    - Check GPU driver: `nvidia-smi`
    - Open Control Panel -> System -> Advanced System Settings -> Environment Variables
    - Verify CUDA paths in System PATH

2.3 Install Qt 6.8.1

1. Download Qt Online Installer
2. Select components:

    - Qt 6.8.1 MSVC 2022 64-bit
    - Qt Debug Information Files
    - Qt Creator
    - CMake

- Ninja

2.4 Build OpenCV 4.10.0

2.4.1 Download Source Files

1. Download OpenCV 4.10.0:
    - Go to: https://opencv.org/releases/
    - Find OpenCV 4.10.0
    - Click "Windows" to download opencv-4.10.0-windows.exe
    - Run the executable to extract files
    - Move the extracted opencv folder to D:/opencv_build/opencv-4.10.0
2. Download OpenCV contrib modules:
    - Go to: https://github.com/opencv/opencv_contrib/releases
    - Download opencv_contrib-4.10.0.zip
    - Extract to D:/opencv_build/opencv_contrib-4.10.0

Note: While OpenCV main library is downloaded from the official releases page, opencv_contrib is still downloaded from GitHub as it's not included in the main release package.

2.4.2 CMake Configuration

1. Create build directory:

```
mkdir D:/opencv_build/build
cd D:/opencv_build/build
```

2. Open CMake GUI:
    - Set source path: D:/opencv_build/opencv-4.10.0
    - Set build path: D:/opencv_build/build
    - Click "Configure"
    - Select "Visual Studio 17 2022" and "x64"
3. Set CMake variables:

```
# Basic Configuration
CMAKE_BUILD_TYPE=Debug
CMAKE_CONFIGURATION_TYPES=Debug
CMAKE_INSTALL_PREFIX=D:/opencv_build/install

# IMPORTANT WARNING ⚠
# DO NOT enable BUILD_opencv_world=ON when building with CUDA support
# It can cause serious linking problems and runtime errors
BUILD_opencv_world=OFF # Keep this OFF for CUDA builds

# Build Options
BUILD_SHARED_LIBS=ON
BUILD_WITH_DEBUG_INFO=ON
BUILD_EXAMPLES=OFF
BUILD_TESTS=OFF
BUILD_PERF_TESTS=OFF
BUILD_JAVA=OFF
BUILD_PACKAGE=OFF

# CUDA Configuration - Set based on your GPU
CUDA_ARCH_BIN=your_value # Set based on your GPU model
CUDA_ARCH_PTX=your_value # Usually same as CUDA_ARCH_BIN

# Enable CUDA
WITH_CUDA=ON
WITH_CUBLAS=ON
CUDA_FAST_MATH=ON
WITH_CUDNN=ON
OPENCV_DNN_CUDA=ON
CUDA_NVCC_FLAGS=--expt-relaxed-constexpr

# Qt Integration
WITH_QT=ON
Qt6_DIR=C:/Qt/6.8.1/msvc2022_64/lib/cmake/Qt6

# OpenCV Contrib Modules
OPENCV_ENABLE_NONFREE=ON
OPENCV_EXTRA_MODULES_PATH=D:/opencv_build/opencv_contrib-4.10.0/modules

# Additional Options
CPU_BASELINE=AVX2
WITH_OPENCL=ON
WITH_OPENGL=ON
ENABLE_CXX11=ON
```

4. Click "Configure" again and check for any errors (marked in red)
5. Click "Generate"

2.4.3 Build OpenCV

1. Open Solution in Visual Studio:

```
cd D:/opencv_build/build
start OpenCV.sln
```

2. In Visual Studio:
    - Set Solution Configuration to "Debug"
    - Set Solution Platform to "x64"
    - Right-click on "ALL_BUILD" project
    - Select "Build"
3. If INSTALL project is grayed out or not activated:
    - Right-click Solution 'OpenCV' in Solution Explorer
    - Select 'Configuration Manager'
    - In Configuration Manager:
        - Make sure 'Debug' is selected
        - Check the 'Build' box for 'INSTALL' project
    - Click 'Close'

    Also verify:
    - Right-click 'INSTALL' project
    - Select 'Set as Startup Project'
    - Right-click 'INSTALL' again
    - Select 'Project Only' -> 'Build Only'
    - Now right-click on "INSTALL" project
    - Select "Build"

Note: If INSTALL is still not visible:

1. Close Visual Studio
2. Delete the CMake cache:
    - Go to D:/opencv_build/build
    - Delete CMakeCache.txt
3. In CMake GUI:
    - Click 'Configure'
    - Verify CMAKE_INSTALL_PREFIX is set to D:/opencv_build/install
    - Click 'Generate'
4. Open OpenCV.sln again
   The INSTALL project should now be available
5. Add to system PATH:

```
setx OPENCV_DIR D:\opencv_build\install
setx PATH "%PATH%;%OPENCV_DIR%\x64\vc17\bin;C:\Program Files\NVIDIA GPU
Computing Toolkit\CUDA\v12.x\bin"
```

3. Project Configuration

Qt Project (.pro) File

```
QT += core gui widgets
CONFIG += c++17
CONFIG += debug

DEFINES += QT_DEPRECATED_WARNINGS

# OpenCV paths
INCLUDEPATH += D:/opencv_build/install/include

LIBS += -LD:/opencv_build/install/x64/vc17/lib \
    -lopencv_core4100d \
    -lopencv_imgproc4100d \
    -lopencv_highgui4100d \
    -lopencv_imgcodecs4100d \
    -lopencv_videoio4100d \
    -lopencv_features2d4100d \
    -lopencv_calib3d4100d \
    -lopencv_objdetect4100d \
    -lopencv_dnn4100d \
    -lopencv_video4100d \
    -lopencv_cudaimgproc4100d \
    -lopencv_cudafilters4100d

SOURCES += \
    main.cpp \
    mainwindow.cpp

HEADERS += \
    mainwindow.h

FORMS += \
    mainwindow.ui
```

4. Required DLLs

Copy these DLLs to your debug folder (e.g., D:\project\build\Desktop_Qt_6_8_1_MSVC2022_64bit-Debug\debug):

From D:/opencv_build/install/x64/vc17/bin:

- opencv_core4100d.dll
- opencv_cudaimgproc4100d.dll
- opencv_imgproc4100d.dll
- opencv_highgui4100d.dll
- opencv_cudafilters4100d.dll
- opencv_imgcodecs4100d.dll
- opencv_videoio4100d.dll

From CUDA installation:

- C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v12.x\bin\cudart64_12.dll

5. Important Warnings

1. DO NOT Enable opencv_world
    - Never set BUILD_opencv_world=ON when using CUDA
    - This can cause:
        - Linking errors during build
        - Runtime crashes
        - DLL loading issues
        - Memory allocation problems
    - Instead, use individual modules as shown in the .pro file configuration
2. Always match CUDA version with your system:
    - Check nvidia-smi output
    - Install matching CUDA version
    - Use compatible cuDNN version
3. Path considerations:
    - Use consistent paths throughout the installation
    - Double-check all environment variables
    - Verify DLL locations in debug folder

6. Test Codes

6.1 Basic CUDA Test

This code verifies basic CUDA functionality:

```
#include <QApplication>
#include <QMessageBox>
#include <opencv2/opencv.hpp>
#include <opencv2/cudaimgproc.hpp>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    try {
        int deviceCount = cv::cuda::getCudaEnabledDeviceCount();
        QMessageBox::information(nullptr, "CUDA Test",
            QString("CUDA Devices Found: %1").arg(deviceCount));
    }
    catch (const cv::Exception& e) {
        QMessageBox::critical(nullptr, "Error",
            QString("CUDA Test Failed: %1").arg(e.what()));
    }
    return a.exec();
}
```

6.2 CUDA Processing Test

This code tests CUDA image processing capabilities:

```
#include <QApplication>
#include <QMessageBox>
#include <opencv2/opencv.hpp>
#include <opencv2/cudaimgproc.hpp>
#include <opencv2/cudafilters.hpp>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    try {
        // Check CUDA device
        int deviceCount = cv::cuda::getCudaEnabledDeviceCount();
        if (deviceCount == 0)
            throw cv::Exception(0, "No CUDA devices found", "", __FILE__,
__LINE__);

        // Create test image
        cv::Mat cpuSrc(1000, 1000, CV_8UC3, cv::Scalar(255, 0, 0));
        cv::cuda::GpuMat gpuSrc;
        gpuSrc.upload(cpuSrc);

        // Apply Gaussian blur using CUDA
        cv::Ptr<cv::cuda::Filter> gaussian = cv::cuda::createGaussianFilter(
            CV_8UC3, CV_8UC3, cv::Size(5, 5), 1.0);
        cv::cuda::GpuMat gpuDst;
        gaussian->apply(gpuSrc, gpuDst);

        // Download result
        cv::Mat cpuDst;
        gpuDst.download(cpuDst);

        QMessageBox::information(nullptr, "Success",
            QString("CUDA Test Passed!\nDevices: %1\nImage Size: %2x%3")
            .arg(deviceCount)
            .arg(cpuDst.cols)
            .arg(cpuDst.rows));
    }
    catch (const cv::Exception& e) {
        QMessageBox::critical(nullptr, "Error",
            QString("CUDA Test Failed: %1").arg(e.what()));
    }
    return a.exec();
}
```

6.3 Image Processing Test

This code processes an actual image file:

```
#include <QApplication>
#include <QMessageBox>
#include <QFileInfo>
#include <opencv2/opencv.hpp>
#include <opencv2/cudaimgproc.hpp>
#include <opencv2/cudafilters.hpp>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    try {
        // Check CUDA device
        int deviceCount = cv::cuda::getCudaEnabledDeviceCount();
        if (deviceCount == 0)
            throw cv::Exception(0, "No CUDA devices found", "", __FILE__,
__LINE__);

        // Read input image using full path
        QString imagePath = "D:/your_project_path/input.jpg";
        if (!QFileInfo::exists(imagePath)) {
            throw cv::Exception(0, "Input image not found: " +
imagePath.toStdString(), "", __FILE__, __LINE__);
        }

        cv::Mat cpuSrc = cv::imread(imagePath.toStdString());
        if (cpuSrc.empty()) {
            throw cv::Exception(0, "Failed to load image: " +
imagePath.toStdString(), "", __FILE__, __LINE__);
        }

        // Upload to GPU
        cv::cuda::GpuMat gpuSrc;
        gpuSrc.upload(cpuSrc);

        // Process on GPU
        cv::cuda::GpuMat gpuGray;
        cv::cuda::cvtColor(gpuSrc, gpuGray, cv::COLOR_BGR2GRAY);

        // Gaussian blur
        cv::Ptr<cv::cuda::Filter> gaussian = cv::cuda::createGaussianFilter(
            CV_8UC1, CV_8UC1, cv::Size(5, 5), 1.0);
        cv::cuda::GpuMat gpuBlurred;
        gaussian->apply(gpuGray, gpuBlurred);

        // Save results
        cv::Mat cpuGray;
        gpuGray.download(cpuGray);
        cv::imwrite("output_gray.jpg", cpuGray);

        cv::Mat cpuBlurred;
        gpuBlurred.download(cpuBlurred);
```

```
        cv::imwrite("output_blur.jpg", cpuBlurred);

        QMessageBox::information(nullptr, "Success",
            QString("CUDA Image Processing Complete!\n"
                    "Input Size: %1x%2\n"
                    "Files saved successfully")
                .arg(cpuSrc.cols)
                .arg(cpuSrc.rows));
    }
    catch (const cv::Exception& e) {
        QMessageBox::critical(nullptr, "Error",
            QString("Processing Failed: %1").arg(e.what()));
    }
    return a.exec();
}
```

7. GPU Compute Capability Reference

RTX 40 Series

GPU Model	Compute Capability
RTX 4090	8.9
RTX 4080	8.9
RTX 4070 Ti	8.9
RTX 4070	8.9

RTX 30 Series

GPU Model	Compute Capability
RTX 3090	8.6
RTX 3080	8.6
RTX 3070	8.6
RTX 3060	8.6

RTX 20 Series

GPU Model	Compute Capability
RTX 2080 Ti	7.5
RTX 2080	7.5
RTX 2070	7.5
RTX 2060	7.5

GTX 16 Series

GPU Model	Compute Capability
GTX 1660 Ti	7.5
GTX 1660	7.5
GTX 1650	7.5

GTX 10 Series

GPU Model	Compute Capability
GTX 1080 Ti	6.1
GTX 1080	6.1
GTX 1070	6.1
GTX 1060	6.1

Quadro/Professional Series

GPU Model	Compute Capability
RTX 6000	8.9
RTX 5000	8.6
RTX 4000	7.5
T2000	7.5

Commands to Verify GPU and CUDA

```
# Check GPU model and CUDA version
nvidia-smi

# Check detailed CUDA info
nvcc --version

# List GPU capabilities (if CUDA samples are installed)
"C:\ProgramData\NVIDIA Corporation\CUDA
Samples\v12.x\bin\x64\Release\deviceQuery.exe"
```

⛔ STOP AND READ BEFORE PROCEEDING ⛔

```
SECTION 8 IS ONLY FOR USERS WHO:
1. Encountered build failures specifically in the cudaimgproc library
2. Have errors related to histogram.cpp in Visual Studio
3. Have already tried all other troubleshooting steps

IF YOU HAVE NOT ENCOUNTERED THESE SPECIFIC ISSUES:
- SKIP SECTION 8
- GO DIRECTLY TO SECTION 9 (TROUBLESHOOTING)

⚠ Modifying core OpenCV files without encountering these specific issues
   may lead to unstable builds and unexpected behavior! ⚠
```

8. Updating OpenCV Contrib Histogram Implementation

Updating histogram.cpp

Navigate to the histogram.cpp file in your OpenCV contrib modules:

```
C:\opencv_contrib-4.x\modules\cudaimgproc\src\histogram.cpp
```

Replace the content with the following code:

```
/*M///////////////////////////////////////////////////////////////////////
///////////
//
//  IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.
//
//  By downloading, copying, installing or using the software you agree to this
license.
//  If you do not agree to this license, do not download, install,
//  copy or use the software.
//
//
//                          License Agreement
//               For Open Source Computer Vision Library
//
// Copyright (C) 2000-2008, Intel Corporation, all rights reserved.
// Copyright (C) 2009, Willow Garage Inc., all rights reserved.
// Third party copyrights are property of their respective owners.
//
// Redistribution and use in source and binary forms, with or without
modification,
// are permitted provided that the following conditions are met:
//
//   * Redistribution's of source code must retain the above copyright notice,
//     this list of conditions and the following disclaimer.
//
//   * Redistribution's in binary form must reproduce the above copyright
notice,
//     this list of conditions and the following disclaimer in the
documentation
//     and/or other materials provided with the distribution.
//
//   * The name of the copyright holders may not be used to endorse or promote
products
//     derived from this software without specific prior written permission.
//
// This software is provided by the copyright holders and contributors "as is"
and
// any express or implied warranties, including, but not limited to, the
implied
// warranties of merchantability and fitness for a particular purpose are
disclaimed.
// In no event shall the Intel Corporation or contributors be liable for any
direct,
// indirect, incidental, special, exemplary, or consequential damages
// (including, but not limited to, procurement of substitute goods or services;
// loss of use, data, or profits; or business interruption) however caused
// and on any theory of liability, whether in contract, strict liability,
// or tort (including negligence or otherwise) arising in any way out of
// the use of this software, even if advised of the possibility of such damage.
//
```

```
//M*/

#include "precomp.hpp"

using namespace cv;
using namespace cv::cuda;

using hist_t = void (*)(const GpuMat&, OutputArray, int, int, int, Stream&);

#if !defined (HAVE_CUDA) || defined (CUDA_DISABLER)

void cv::cuda::calcHist(InputArray, OutputArray, Stream&) { throw_no_cuda(); }
void cv::cuda::calcHist(InputArray, InputArray, OutputArray, Stream&) {
throw_no_cuda(); }

void cv::cuda::equalizeHist(InputArray, OutputArray, Stream&) {
throw_no_cuda(); }

cv::Ptr<cv::cuda::CLAHE> cv::cuda::createCLAHE(double, cv::Size) {
throw_no_cuda(); return cv::Ptr<cv::cuda::CLAHE>(); }

void cv::cuda::evenLevels(OutputArray, int, int, int, Stream&) {
throw_no_cuda(); }

void cv::cuda::histEven(InputArray, OutputArray, int, int, int, Stream&) {
throw_no_cuda(); }
void cv::cuda::histEven(InputArray, GpuMat*, int*, int*, int*, Stream&) {
throw_no_cuda(); }

void cv::cuda::histRange(InputArray, OutputArray, InputArray, Stream&) {
throw_no_cuda(); }
void cv::cuda::histRange(InputArray, GpuMat*, const GpuMat*, Stream&) {
throw_no_cuda(); }

#else /* !defined (HAVE_CUDA) */

////////////////////////////////////////////////////////////////////////
// calcHist

namespace hist
{
    void histogram256(PtrStepSzb src, int* hist, const int offsetX,
cudaStream_t stream);
    void histogram256(PtrStepSzb src, PtrStepSzb mask, int* hist, const int
offsetX, cudaStream_t stream);
}

void cv::cuda::calcHist(InputArray _src, OutputArray _hist, Stream& stream)
{
    calcHist(_src, cv::cuda::GpuMat(), _hist, stream);
}
```

```
void cv::cuda::calcHist(InputArray _src, InputArray _mask, OutputArray _hist,
Stream& stream)
{
    GpuMat src = _src.getGpuMat();
    GpuMat mask = _mask.getGpuMat();

    CV_Assert(src.type() == CV_8UC1);
    CV_Assert(mask.empty() || mask.type() == CV_8UC1);
    CV_Assert(mask.empty() || mask.size() == src.size());

    _hist.create(1, 256, CV_32SC1);
    GpuMat hist = _hist.getGpuMat();

    hist.setTo(Scalar::all(0), stream);

    Point ofs; Size wholeSize;
    src.locateROI(wholeSize, ofs);
    if (mask.empty())
        hist::histogram256(src, hist.ptr<int>(), ofs.x,
StreamAccessor::getStream(stream));
    else
        hist::histogram256(src, mask, hist.ptr<int>(), ofs.x,
StreamAccessor::getStream(stream));
}

////////////////////////////////////////////////////////////////////////
// equalizeHist

namespace hist
{
    void equalizeHist(PtrStepSzb src, PtrStepSzb dst, const uchar* lut,
cudaStream_t stream);
    void buildLut(PtrStepSzi hist, PtrStepSzb lut, int size, cudaStream_t
stream);
}

void cv::cuda::equalizeHist(InputArray _src, OutputArray _dst, Stream& _stream)
{
    GpuMat src = getInputMat(_src, _stream);

    CV_Assert(src.type() == CV_8UC1);

    _dst.create(src.size(), src.type());
    GpuMat dst = _dst.getGpuMat();

    size_t bufSize = 256 * sizeof(int) + 256 * sizeof(uchar);

    BufferPool pool(_stream);
    GpuMat buf = pool.getBuffer(1, static_cast<int>(bufSize), CV_8UC1);

    GpuMat hist(1, 256, CV_32SC1, buf.data);
    GpuMat lut(1, 256, CV_8UC1, buf.data + 256 * sizeof(int));
```

```
    cuda::calcHist(src, hist, _stream);

    cudaStream_t stream = StreamAccessor::getStream(_stream);

    hist::buildLut(hist, lut, src.rows * src.cols, stream);

    hist::equalizeHist(src, dst, lut.data, stream);
}

////////////////////////////////////////////////////////////////////////
// CLAHE

namespace clahe
{
    void calcLut_8U(PtrStepSzb src, PtrStepb lut, int tilesX, int tilesY, int2
tileSize, int clipLimit, float lutScale, cudaStream_t stream);
    void calcLut_16U(PtrStepSzus src, PtrStepus lut, int tilesX, int tilesY,
int2 tileSize, int clipLimit, float lutScale, PtrStepSzi hist, cudaStream_t
stream);
    template <typename T> void transform(PtrStepSz<T> src, PtrStepSz<T> dst,
PtrStep<T> lut, int tilesX, int tilesY, int2 tileSize, cudaStream_t stream);
}

namespace
{
    class CLAHE_Impl : public cv::cuda::CLAHE
    {
    public:
        CLAHE_Impl(double clipLimit = 40.0, int tilesX = 8, int tilesY = 8);

        void apply(cv::InputArray src, cv::OutputArray dst);
        void apply(InputArray src, OutputArray dst, Stream& stream);

        void setClipLimit(double clipLimit);
        double getClipLimit() const;

        void setTilesGridSize(cv::Size tileGridSize);
        cv::Size getTilesGridSize() const;

        void collectGarbage();

    private:
        double clipLimit_;
        int tilesX_;
        int tilesY_;

        GpuMat srcExt_;
        GpuMat lut_;
        GpuMat hist_; // histogram on global memory for CV_16UC1 case
    };
```

```
    CLAHE_Impl::CLAHE_Impl(double clipLimit, int tilesX, int tilesY) :
        clipLimit_(clipLimit), tilesX_(tilesX), tilesY_(tilesY)
    {
    }

    void CLAHE_Impl::apply(cv::InputArray _src, cv::OutputArray _dst)
    {
        apply(_src, _dst, Stream::Null());
    }

    void CLAHE_Impl::apply(InputArray _src, OutputArray _dst, Stream& s)
    {
        GpuMat src = _src.getGpuMat();

        const int type = src.type();

        CV_Assert(type == CV_8UC1 || type == CV_16UC1);

        _dst.create(src.size(), type);
        GpuMat dst = _dst.getGpuMat();

        const int histSize = type == CV_8UC1 ? 256 : 65536;

        ensureSizeIsEnough(tilesX_ * tilesY_, histSize, type, lut_);

        cudaStream_t stream = StreamAccessor::getStream(s);

        cv::Size tileSize;
        GpuMat srcForLut;

        if (src.cols % tilesX_ == 0 && src.rows % tilesY_ == 0)
        {
            tileSize = cv::Size(src.cols / tilesX_, src.rows / tilesY_);
            srcForLut = src;
        }
        else
        {
#ifndef HAVE_OPENCV_CUDAARITHM
            throw_no_cuda();
#else
            cv::cuda::copyMakeBorder(src, srcExt_, 0, tilesY_ - (src.rows %
tilesY_), 0, tilesX_ - (src.cols % tilesX_), cv::BORDER_REFLECT_101,
cv::Scalar(), s);
#endif

            tileSize = cv::Size(srcExt_.cols / tilesX_, srcExt_.rows /
tilesY_);
            srcForLut = srcExt_;
        }

        const int tileSizeTotal = tileSize.area();
        const float lutScale = static_cast<float>(histSize - 1) /
```

```
tileSizeTotal;

        int clipLimit = 0;
        if (clipLimit_ > 0.0)
        {
            clipLimit = static_cast<int>(clipLimit_ * tileSizeTotal /
histSize);
            clipLimit = std::max(clipLimit, 1);
        }

        if (type == CV_8UC1)
            clahe::calcLut_8U(srcForLut, lut_, tilesX_, tilesY_,
make_int2(tileSize.width, tileSize.height), clipLimit, lutScale, stream);
        else // type == CV_16UC1
        {
            ensureSizeIsEnough(tilesX_ * tilesY_, histSize, CV_32SC1, hist_);
            clahe::calcLut_16U(srcForLut, lut_, tilesX_, tilesY_,
make_int2(tileSize.width, tileSize.height), clipLimit, lutScale, hist_,
stream);
        }

        if (type == CV_8UC1)
            clahe::transform<uchar>(src, dst, lut_, tilesX_, tilesY_,
make_int2(tileSize.width, tileSize.height), stream);
        else // type == CV_16UC1
            clahe::transform<ushort>(src, dst, lut_, tilesX_, tilesY_,
make_int2(tileSize.width, tileSize.height), stream);
    }

    void CLAHE_Impl::setClipLimit(double clipLimit)
    {
        clipLimit_ = clipLimit;
    }

    double CLAHE_Impl::getClipLimit() const
    {
        return clipLimit_;
    }

    void CLAHE_Impl::setTilesGridSize(cv::Size tileGridSize)
    {
        tilesX_ = tileGridSize.width;
        tilesY_ = tileGridSize.height;
    }

    cv::Size CLAHE_Impl::getTilesGridSize() const
    {
        return cv::Size(tilesX_, tilesY_);
    }

    void CLAHE_Impl::collectGarbage()
    {
```

```
        srcExt_.release();
        lut_.release();
    }
}

cv::Ptr<cv::cuda::CLAHE> cv::cuda::createCLAHE(double clipLimit, cv::Size
tileGridSize)
{
    return makePtr<CLAHE_Impl>(clipLimit, tileGridSize.width,
tileGridSize.height);
}

////////////////////////////////////////////////////////////////////////
// NPP Histogram

namespace {

#if (NPP_VERSION >= 12205)
    // Implementation for 8-bit single channel
    NppStatus getBufSize8uC1(NppiSize size, int levels, size_t* bufferSize,
NppStreamContext ctx) {
        return nppiHistogramEvenGetBufferSize_8u_C1R_Ctx(size, levels,
bufferSize, ctx);
    }

    // Implementation for 16-bit unsigned single channel
    NppStatus getBufSize16uC1(NppiSize size, int levels, size_t* bufferSize,
NppStreamContext ctx) {
        return nppiHistogramEvenGetBufferSize_16u_C1R_Ctx(size, levels,
bufferSize, ctx);
    }

    // Implementation for 16-bit signed single channel
    NppStatus getBufSize16sC1(NppiSize size, int levels, size_t* bufferSize,
NppStreamContext ctx) {
        return nppiHistogramEvenGetBufferSize_16s_C1R_Ctx(size, levels,
bufferSize, ctx);
    }

    // Implementation for 8-bit four channel
    NppStatus getBufSize8uC4(NppiSize size, int levels[], size_t* bufferSize,
NppStreamContext ctx) {
        return nppiHistogramEvenGetBufferSize_8u_C4R_Ctx(size, levels,
bufferSize, ctx);
    }

    // Implementation for 16-bit unsigned four channel
    NppStatus getBufSize16uC4(NppiSize size, int levels[], size_t* bufferSize,
NppStreamContext ctx) {
        return nppiHistogramEvenGetBufferSize_16u_C4R_Ctx(size, levels,
bufferSize, ctx);
    }
```

```
    // Implementation for 16-bit signed four channel
    NppStatus getBufSize16sC4(NppiSize size, int levels[], size_t* bufferSize,
NppStreamContext ctx) {
        return nppiHistogramEvenGetBufferSize_16s_C4R_Ctx(size, levels,
bufferSize, ctx);
    }

#else // NPP_VERSION < 12205

    // Legacy implementations for older NPP versions
    NppStatus getBufSize8uC1(NppiSize size, int levels, int* bufferSize) {
        return nppiHistogramEvenGetBufferSize_8u_C1R(size, levels, bufferSize);
    }

    NppStatus getBufSize16uC1(NppiSize size, int levels, int* bufferSize) {
        return nppiHistogramEvenGetBufferSize_16u_C1R(size, levels,
bufferSize);
    }

    NppStatus getBufSize16sC1(NppiSize size, int levels, int* bufferSize) {
        return nppiHistogramEvenGetBufferSize_16s_C1R(size, levels,
bufferSize);
    }

    NppStatus getBufSize8uC4(NppiSize size, int levels[], int* bufferSize) {
        return nppiHistogramEvenGetBufferSize_8u_C4R(size, levels, bufferSize);
    }

    NppStatus getBufSize16uC4(NppiSize size, int levels[], int* bufferSize) {
        return nppiHistogramEvenGetBufferSize_16u_C4R(size, levels,
bufferSize);
    }

    NppStatus getBufSize16sC4(NppiSize size, int levels[], int* bufferSize) {
        return nppiHistogramEvenGetBufferSize_16s_C4R(size, levels,
bufferSize);
    }

#endif // NPP_VERSION >= 12205

} // anonymous namespace

namespace {
#if (NPP_VERSION >= 12205)
    using get_buf_size_c1_t = NppStatus(*)(NppiSize, int, size_t*,
NppStreamContext);
    using get_buf_size_c4_t = NppStatus(*)(NppiSize, int[], size_t*,
NppStreamContext);
#else
    using get_buf_size_c1_t = NppStatus(*)(NppiSize, int, int*);
    using get_buf_size_c4_t = NppStatus(*)(NppiSize, int[], int*);
```

```
#endif

    using hist_t = void (*)(const GpuMat&, OutputArray, int, int, int,
Stream&);
    using hist_c4_t = void (*)(const GpuMat&, GpuMat[4], int[4], int[4],
int[4], Stream&);

    // Base templates for histogram function types
    template<int SDEPTH>
    struct NppHistogramEvenFuncC1
    {
        typedef typename NPPTypeTraits<SDEPTH>::npp_type src_t;
#if (NPP_VERSION >= 12205)
        typedef NppStatus(*func_ptr)(const src_t*, int, NppiSize, Npp32s*, int,
Npp32s, Npp32s, Npp8u*, NppStreamContext);
#else
        typedef NppStatus(*func_ptr)(const src_t*, int, NppiSize, Npp32s*, int,
Npp32s, Npp32s, Npp8u*);
#endif
    };

    template<int SDEPTH>
    struct NppHistogramEvenFuncC4
    {
        typedef typename NPPTypeTraits<SDEPTH>::npp_type src_t;
#if (NPP_VERSION >= 12205)
        typedef NppStatus(*func_ptr)(const src_t*, int, NppiSize, Npp32s* [4],
int[4], Npp32s[4], Npp32s[4], Npp8u*, NppStreamContext);
#else
        typedef NppStatus(*func_ptr)(const src_t*, int, NppiSize, Npp32s* [4],
int[4], Npp32s[4], Npp32s[4], Npp8u*);
#endif
    };

    // Helper functions forward declarations
#if (NPP_VERSION >= 12205)
    NppStatus getBufSize8uC1(NppiSize size, int levels, size_t* bufferSize,
NppStreamContext ctx);
    NppStatus getBufSize16uC1(NppiSize size, int levels, size_t* bufferSize,
NppStreamContext ctx);
    NppStatus getBufSize16sC1(NppiSize size, int levels, size_t* bufferSize,
NppStreamContext ctx);
    NppStatus getBufSize8uC4(NppiSize size, int levels[], size_t* bufferSize,
NppStreamContext ctx);
    NppStatus getBufSize16uC4(NppiSize size, int levels[], size_t* bufferSize,
NppStreamContext ctx);
    NppStatus getBufSize16sC4(NppiSize size, int levels[], size_t* bufferSize,
NppStreamContext ctx);
#else
    NppStatus getBufSize8uC1(NppiSize size, int levels, int* bufferSize);
    NppStatus getBufSize16uC1(NppiSize size, int levels, int* bufferSize);
    NppStatus getBufSize16sC1(NppiSize size, int levels, int* bufferSize);
```

```
    NppStatus getBufSize8uC4(NppiSize size, int levels[], int* bufferSize);
    NppStatus getBufSize16uC4(NppiSize size, int levels[], int* bufferSize);
    NppStatus getBufSize16sC4(NppiSize size, int levels[], int* bufferSize);
#endif

    // NPP function pointers forward declarations
#if (NPP_VERSION >= 12205)
    extern const typename NppHistogramEvenFuncC1<CV_8U>::func_ptr
nppiHistogramEven_8u_C1R_p;
    extern const typename NppHistogramEvenFuncC1<CV_16U>::func_ptr
nppiHistogramEven_16u_C1R_p;
    extern const typename NppHistogramEvenFuncC1<CV_16S>::func_ptr
nppiHistogramEven_16s_C1R_p;
    extern const typename NppHistogramEvenFuncC4<CV_8U>::func_ptr
nppiHistogramEven_8u_C4R_p;
    extern const typename NppHistogramEvenFuncC4<CV_16U>::func_ptr
nppiHistogramEven_16u_C4R_p;
    extern const typename NppHistogramEvenFuncC4<CV_16S>::func_ptr
nppiHistogramEven_16s_C4R_p;
#else
    extern const typename NppHistogramEvenFuncC1<CV_8U>::func_ptr
nppiHistogramEven_8u_C1R;
    extern const typename NppHistogramEvenFuncC1<CV_16U>::func_ptr
nppiHistogramEven_16u_C1R;
    extern const typename NppHistogramEvenFuncC1<CV_16S>::func_ptr
nppiHistogramEven_16s_C1R;
    extern const typename NppHistogramEvenFuncC4<CV_8U>::func_ptr
nppiHistogramEven_8u_C4R;
    extern const typename NppHistogramEvenFuncC4<CV_16U>::func_ptr
nppiHistogramEven_16u_C4R;
    extern const typename NppHistogramEvenFuncC4<CV_16S>::func_ptr
nppiHistogramEven_16s_C4R;
#endif

    // Implementation templates
    template<int SDEPTH, typename NppHistogramEvenFuncC1<SDEPTH>::func_ptr
func, get_buf_size_c1_t get_buf_size>
    struct NppHistogramEvenC1
    {
        typedef typename NppHistogramEvenFuncC1<SDEPTH>::src_t src_t;

        static void hist(const GpuMat& src, OutputArray _hist, int histSize,
int lowerLevel, int upperLevel, Stream& stream)
        {
            const int levels = histSize + 1;
            _hist.create(1, histSize, CV_32S);
            GpuMat hist = _hist.getGpuMat();

            NppiSize sz;
            sz.width = src.cols;
            sz.height = src.rows;
```

```
            NppStreamHandler h(StreamAccessor::getStream(stream));

#if (NPP_VERSION >= 12205)
            size_t buf_size = 0;
            NppStreamContext nppStreamCtx = h.nppStreamCtx();  // Get proper
NPP stream context
            nppSafeCall(get_buf_size(sz, levels, &buf_size, nppStreamCtx));

            BufferPool pool(stream);
            CV_Assert(buf_size <= std::numeric_limits<int>::max());
            GpuMat buf = pool.getBuffer(1, static_cast<int>(buf_size),
CV_8UC1);

            nppSafeCall(func(src.ptr<src_t>(), static_cast<int>(src.step), sz,
hist.ptr<Npp32s>(),
                levels, lowerLevel, upperLevel, buf.ptr<Npp8u>(),
nppStreamCtx));
#else
            int buf_size = 0;
            nppSafeCall(get_buf_size(sz, levels, &buf_size));

            BufferPool pool(stream);
            GpuMat buf = pool.getBuffer(1, buf_size, CV_8UC1);

            nppSafeCall(func(src.ptr<src_t>(), static_cast<int>(src.step), sz,
hist.ptr<Npp32s>(),
                levels, lowerLevel, upperLevel, buf.ptr<Npp8u>()));
#endif

            if (!stream)
                cudaSafeCall(cudaDeviceSynchronize());
        }
    };

    template<int SDEPTH, typename NppHistogramEvenFuncC4<SDEPTH>::func_ptr
func, get_buf_size_c4_t get_buf_size>
    struct NppHistogramEvenC4
    {
        typedef typename NppHistogramEvenFuncC4<SDEPTH>::src_t src_t;

        static void hist(const GpuMat& src, GpuMat hist[4], int histSize[4],
int lowerLevel[4], int upperLevel[4], Stream& stream)
        {
            int levels[] = { histSize[0] + 1, histSize[1] + 1, histSize[2] + 1,
histSize[3] + 1 };

            for (int i = 0; i < 4; i++)
                hist[i].create(1, histSize[i], CV_32S);

            NppiSize sz;
            sz.width = src.cols;
            sz.height = src.rows;
```

```
            Npp32s* pHist[] = { hist[0].ptr<Npp32s>(), hist[1].ptr<Npp32s>(),
hist[2].ptr<Npp32s>(), hist[3].ptr<Npp32s>() };

            NppStreamHandler h(StreamAccessor::getStream(stream));

#if (NPP_VERSION >= 12205)
            size_t buf_size = 0;
            NppStreamContext nppStreamCtx = h.nppStreamCtx();  // Get proper
NPP stream context
            nppSafeCall(get_buf_size(sz, levels, &buf_size, nppStreamCtx));
#else
            int buf_size = 0;
            nppSafeCall(get_buf_size(sz, levels, &buf_size));
#endif

            BufferPool pool(stream);
            CV_Assert(buf_size <= std::numeric_limits<int>::max());
            GpuMat buf = pool.getBuffer(1, static_cast<int>(buf_size),
CV_8UC1);

#if (NPP_VERSION >= 12205)
            nppSafeCall(func(src.ptr<src_t>(), static_cast<int>(src.step), sz,
pHist,
                levels, lowerLevel, upperLevel, buf.ptr<Npp8u>(),
nppStreamCtx));
#else
            nppSafeCall(func(src.ptr<src_t>(), static_cast<int>(src.step), sz,
pHist,
                levels, lowerLevel, upperLevel, buf.ptr<Npp8u>()));
#endif

            if (!stream)
                cudaSafeCall(cudaDeviceSynchronize());
        }
    };

    // Function pointers for histogram calculation
    static struct HistCallers {
        hist_t funcs[4];
        hist_c4_t funcs_c4[4];

        HistCallers() {
            // Initialize all pointers to nullptr first
            for (int i = 0; i < 4; i++) {
                funcs[i] = nullptr;
                funcs_c4[i] = nullptr;
            }

#if (NPP_VERSION >= 12205)
            funcs[CV_8U] = NppHistogramEvenC1<CV_8U,
nppiHistogramEven_8u_C1R_Ctx, getBufSize8uC1>::hist;
```

```
            funcs[CV_16U] = NppHistogramEvenC1<CV_16U,
nppiHistogramEven_16u_C1R_Ctx, getBufSize16uC1>::hist;
            funcs[CV_16S] = NppHistogramEvenC1<CV_16S,
nppiHistogramEven_16s_C1R_Ctx, getBufSize16sC1>::hist;

            funcs_c4[CV_8U] = NppHistogramEvenC4<CV_8U,
nppiHistogramEven_8u_C4R_Ctx, getBufSize8uC4>::hist;
            funcs_c4[CV_16U] = NppHistogramEvenC4<CV_16U,
nppiHistogramEven_16u_C4R_Ctx, getBufSize16uC4>::hist;
            funcs_c4[CV_16S] = NppHistogramEvenC4<CV_16S,
nppiHistogramEven_16s_C4R_Ctx, getBufSize16sC4>::hist;
#else
            funcs[CV_8U] = NppHistogramEvenC1<CV_8U, nppiHistogramEven_8u_C1R,
getBufSize8uC1>::hist;
            funcs[CV_16U] = NppHistogramEvenC1<CV_16U,
nppiHistogramEven_16u_C1R, getBufSize16uC1>::hist;
            funcs[CV_16S] = NppHistogramEvenC1<CV_16S,
nppiHistogramEven_16s_C1R, getBufSize16sC1>::hist;

            funcs_c4[CV_8U] = NppHistogramEvenC4<CV_8U,
nppiHistogramEven_8u_C4R, getBufSize8uC4>::hist;
            funcs_c4[CV_16U] = NppHistogramEvenC4<CV_16U,
nppiHistogramEven_16u_C4R, getBufSize16uC4>::hist;
            funcs_c4[CV_16S] = NppHistogramEvenC4<CV_16S,
nppiHistogramEven_16s_C4R, getBufSize16sC4>::hist;
#endif
        }

        hist_t operator[](int depth) {
            return (depth >= 0 && depth < 4) ? funcs[depth] : nullptr;
        }

        hist_c4_t get_c4(int depth) {
            return (depth >= 0 && depth < 4) ? funcs_c4[depth] : nullptr;
        }
    } hist_callers;
}

// Place this before the cv::cuda::histEven functions

namespace hist
{
    void histEven8u(PtrStepSzb src, int* hist, int binCount, int lowerLevel,
int upperLevel, const int offsetX, cudaStream_t stream);
}

namespace
{
    void histEven8uImpl(const GpuMat& src, OutputArray _hist, int histSize, int
lowerLevel, int upperLevel, cudaStream_t stream)
    {
        Point ofs;
```

```
        Size wholeSize;
        src.locateROI(wholeSize, ofs);

        GpuMat hist;
        if (_hist.isGpuMat())
            hist = _hist.getGpuMatRef();
        else
            hist = GpuMat(1, histSize, CV_32S);

        hist.create(1, histSize, CV_32S);
        cudaSafeCall(cudaMemsetAsync(hist.data, 0, histSize * sizeof(int),
stream));
        hist::histEven8u(src, hist.ptr<int>(), histSize, lowerLevel,
upperLevel, ofs.x, stream);

        if (!_hist.isGpuMat())
            hist.download(_hist);
    }
}

void cv::cuda::histEven(InputArray _src, OutputArray hist, int histSize, int
lowerLevel, int upperLevel, Stream& stream)
{
    GpuMat src = _src.getGpuMat();

    if (src.depth() == CV_8U && deviceSupports(FEATURE_SET_COMPUTE_30))
    {
        histEven8uImpl(src, hist, histSize, lowerLevel, upperLevel,
StreamAccessor::getStream(stream));
        return;
    }

    CV_Assert(src.type() == CV_8UC1 || src.type() == CV_16UC1 || src.type() ==
CV_16SC1);

    hist_t func = hist_callers[src.depth()];
    if (func)
        func(src, hist, histSize, lowerLevel, upperLevel, stream);
    else
        CV_Error(Error::StsUnsupportedFormat, "Unsupported depth");
}

void cv::cuda::histEven(InputArray _src, GpuMat hist[4], int histSize[4], int
lowerLevel[4], int upperLevel[4], Stream& stream)
{
    GpuMat src = _src.getGpuMat();

    CV_Assert(src.type() == CV_8UC4 || src.type() == CV_16UC4 || src.type() ==
CV_16SC4);

    hist_c4_t func = hist_callers.get_c4(src.depth());
    if (func)
```

```
        func(src, hist, histSize, lowerLevel, upperLevel, stream);
    else
        CV_Error(Error::StsUnsupportedFormat, "Unsupported depth");
}
#endif /* !defined (HAVE_CUDA) */
```

Updating private.cuda.hpp

Location

Navigate to the private.cuda.hpp file in your OpenCV core modules:

```
C:\opencv-4.10.0\modules\core\include\opencv2\core\private.cuda.hpp
```

Implementation

Update the following code at the appropriate section in private.cuda.hpp:

```
/*M///////////////////////////////////////////////////////////////////////////////////////
//
//  IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING.
//
//  By downloading, copying, installing or using the software you agree to this
license.
//  If you do not agree to this license, do not download, install,
//  copy or use the software.
//
//
//                          License Agreement
//               For Open Source Computer Vision Library
//
// Copyright (C) 2000-2008, Intel Corporation, all rights reserved.
// Copyright (C) 2009, Willow Garage Inc., all rights reserved.
// Copyright (C) 2013, OpenCV Foundation, all rights reserved.
// Third party copyrights are property of their respective owners.
//
// Redistribution and use in source and binary forms, with or without
modification,
// are permitted provided that the following conditions are met:
//
//   * Redistribution's of source code must retain the above copyright notice,
//     this list of conditions and the following disclaimer.
//
//   * Redistribution's in binary form must reproduce the above copyright
notice,
//     this list of conditions and the following disclaimer in the
documentation
//     and/or other materials provided with the distribution.
//
//   * The name of the copyright holders may not be used to endorse or promote
```

```
products
//     derived from this software without specific prior written permission.
//
// This software is provided by the copyright holders and contributors "as is"
and
// any express or implied warranties, including, but not limited to, the
implied
// warranties of merchantability and fitness for a particular purpose are
disclaimed.
// In no event shall the Intel Corporation or contributors be liable for any
direct,
// indirect, incidental, special, exemplary, or consequential damages
// (including, but not limited to, procurement of substitute goods or services;
// loss of use, data, or profits; or business interruption) however caused
// and on any theory of liability, whether in contract, strict liability,
// or tort (including negligence or otherwise) arising in any way out of
// the use of this software, even if advised of the possibility of such damage.
//
//M*/

#ifndef OPENCV_CORE_PRIVATE_CUDA_HPP
#define OPENCV_CORE_PRIVATE_CUDA_HPP

#ifndef __OPENCV_BUILD
#  error this is a private header which should not be used from outside of the
OpenCV library
#endif

#include "cvconfig.h"

#include "opencv2/core/cvdef.h"
#include "opencv2/core/base.hpp"

#include "opencv2/core/cuda.hpp"

#ifdef HAVE_CUDA
#  include <cuda.h>
#  include <cuda_runtime.h>
#  if defined(__CUDACC_VER_MAJOR__) && (8 <= __CUDACC_VER_MAJOR__)
#    if defined (__GNUC__) && !defined(__CUDACC__)
#     pragma GCC diagnostic push
#     pragma GCC diagnostic ignored "-Wstrict-aliasing"
#     include <cuda_fp16.h>
#     pragma GCC diagnostic pop
#    else
#     include <cuda_fp16.h>
#    endif
#  endif // defined(__CUDACC_VER_MAJOR__) && (8 <= __CUDACC_VER_MAJOR__)
#  include <npp.h>
#  include "opencv2/core/cuda_stream_accessor.hpp"
#  include "opencv2/core/cuda/common.hpp"
```

```
# ifndef NPP_VERSION
#  define NPP_VERSION (NPP_VERSION_MAJOR * 1000 + NPP_VERSION_MINOR * 100 +
NPP_VERSION_BUILD)
# endif

#  define CUDART_MINIMUM_REQUIRED_VERSION 6050

#  if (CUDART_VERSION < CUDART_MINIMUM_REQUIRED_VERSION)
#    error "Insufficient Cuda Runtime library version, please update it."
#  endif

#endif

//! @cond IGNORED

namespace cv { namespace cuda {
    CV_EXPORTS cv::String getNppErrorMessage(int code);
    CV_EXPORTS cv::String getCudaDriverApiErrorMessage(int code);

    CV_EXPORTS GpuMat getInputMat(InputArray _src, Stream& stream);

    CV_EXPORTS GpuMat getOutputMat(OutputArray _dst, int rows, int cols, int
type, Stream& stream);
    static inline GpuMat getOutputMat(OutputArray _dst, Size size, int type,
Stream& stream)
    {
        return getOutputMat(_dst, size.height, size.width, type, stream);
    }

    CV_EXPORTS void syncOutput(const GpuMat& dst, OutputArray _dst, Stream&
stream);
}}

#ifndef HAVE_CUDA

static inline CV_NORETURN void throw_no_cuda() {
CV_Error(cv::Error::GpuNotSupported, "The library is compiled without CUDA
support"); }

#else // HAVE_CUDA

#define nppSafeSetStream(oldStream, newStream) { if(oldStream != newStream) {
cudaStreamSynchronize(oldStream); nppSetStream(newStream); } }

static inline CV_NORETURN void throw_no_cuda() {
CV_Error(cv::Error::StsNotImplemented, "The called functionality is disabled
for current build or platform"); }

namespace cv { namespace cuda
{
    static inline void checkNppError(int code, const char* file, const int
line, const char* func)
```

```
    {
        if (code < 0)
            cv::error(cv::Error::GpuApiCallError, getNppErrorMessage(code),
func, file, line);
    }

    static inline void checkCudaDriverApiError(int code, const char* file,
const int line, const char* func)
    {
        if (code != CUDA_SUCCESS)
            cv::error(cv::Error::GpuApiCallError,
getCudaDriverApiErrorMessage(code), func, file, line);
    }

    template<int n> struct NPPTypeTraits;
    template<> struct NPPTypeTraits<CV_8U>  { typedef Npp8u npp_type; };
    template<> struct NPPTypeTraits<CV_8S>  { typedef Npp8s npp_type; };
    template<> struct NPPTypeTraits<CV_16U> { typedef Npp16u npp_type; };
    template<> struct NPPTypeTraits<CV_16S> { typedef Npp16s npp_type; };
    template<> struct NPPTypeTraits<CV_32S> { typedef Npp32s npp_type; };
    template<> struct NPPTypeTraits<CV_32F> { typedef Npp32f npp_type; };
    template<> struct NPPTypeTraits<CV_64F> { typedef Npp64f npp_type; };

    class NppStreamHandler
    {
    public:
        inline explicit NppStreamHandler(Stream& newStream)
        {
            oldStream = nppGetStream();
            nppSafeSetStream(oldStream, StreamAccessor::getStream(newStream));
        }

        inline explicit NppStreamHandler(cudaStream_t newStream)
        {
            oldStream = nppGetStream();
            nppSafeSetStream(oldStream, newStream);
        }

        inline ~NppStreamHandler()
        {
            nppSafeSetStream(nppGetStream(), oldStream);
        }

        NppStreamContext nppStreamCtx() const
        {
            NppStreamContext ctx = { 0 };
            ctx.hStream = stream_;
            ctx.nCudaDeviceId = deviceId_;
            // Set other required NPP stream context properties
            return ctx;
        }
```

```
    private:
        cudaStream_t oldStream;
        cudaStream_t stream_;
        int deviceId_;
    };
}}

#define nppSafeCall(expr)  cv::cuda::checkNppError(expr, __FILE__, __LINE__,
CV_Func)
#define cuSafeCall(expr)  cv::cuda::checkCudaDriverApiError(expr, __FILE__,
__LINE__, CV_Func)

#endif // HAVE_CUDA

//! @endcond

#endif // OPENCV_CORE_PRIVATE_CUDA_HPP
```

9. Troubleshooting Guide

Common Issues and Solutions

1. CUDA Device Not Found
    - Verify GPU driver installation
    - Check nvidia-smi output
    - Ensure CUDA paths are correct in PATH
    - Rebuild OpenCV with correct CUDA configuration
2. Linking Errors
    - Check all required DLLs are in debug folder
    - Verify library versions match
    - Make sure CUDA_ARCH_BIN matches your GPU
    - Clean and rebuild project
3. Runtime Errors
    - Verify all DLLs are present
    - Check CUDA driver version
    - Monitor GPU memory usage
    - Enable debug output for OpenCV
4. Build Configuration Issues
    - Delete CMake cache and reconfigure
    - Check Visual Studio platform settings
    - Verify Qt and OpenCV paths
    - Check environment variables

Verification Steps

1. Run Basic CUDA Test first
2. Check all required DLLs
3. Monitor GPU usage during tests
4. Verify memory allocation
5. Check build configurations match (Debug/Release)