```
#include "opencv2/core/cuda.hpp"
#include "opencv2/calib3d.hpp"

/**
   @addtogroup cuda
   @{
      @defgroup cudastereo Stereo Correspondence
   @}
 */

namespace cv { namespace cuda {

//! @addtogroup cudastereo
//! @{

//////////////////////////////////////
// StereoBM

/** @brief Class computing stereo correspondence (disparity map) using the block matching algorithm. :

@sa StereoBM
 */
class CV_EXPORTS_W StereoBM : public cv::StereoBM
{
public:
      using cv::StereoBM::compute;

      CV_WRAP virtual void compute(InputArray left, InputArray right, OutputArray disparity, Stream& stream) = 0;
};

/** @brief Creates StereoBM object.

@param numDisparities the disparity search range. For each pixel algorithm will find the best disparity from 0 (default minimum disparity) to numDisparities. The search range can then be shifted by changing the minimum disparity.
@param blockSize the linear size of the blocks compared by the algorithm. The size should be odd (as the block is centered at the current pixel). Larger block size implies smoother, though less accurate disparity map. Smaller block size gives more detailed disparity map, but there is higher chance for algorithm to find a wrong correspondence.
```

```
 */
CV_EXPORTS_W Ptr<cuda::StereoBM> createStereoBM(int numDisparities = 64, int blockSize = 19);

//////////////////////////////////////
// StereoBeliefPropagation

/** @brief Class computing stereo correspondence using the belief propagation algorithm. :

The class implements algorithm described in @cite Felzenszwalb2006 . It can compute own data cost
(using a truncated linear model) or use a user-provided data cost.

@note
    StereoBeliefPropagation requires a lot of memory for message storage:

        \f[width \_ step    \cdot height    \cdot ndisp    \cdot 4    \cdot (1 + 0.25)\f]

        and for data cost storage:

        \f[width\_step \cdot height \cdot ndisp \cdot (1 + 0.25 + 0.0625 +      \dotsm +
\frac{1}{4^{levels}})\f]

        width_step is the number of bytes in a line including padding.

StereoBeliefPropagation uses a truncated linear model for the data cost and discontinuity terms:

\f[DataCost = data \_ weight    \cdot \min ( \lvert Img_Left(x,y)-Img_Right(x-d,y)    \rvert , max \_
data \_ term)\f]

\f[DiscTerm =    \min (disc \_ single \_ jump    \cdot \lvert f_1-f_2    \rvert , max \_ disc \_ term)\f]

For more details, see @cite Felzenszwalb2006 .

By default, StereoBeliefPropagation uses floating-point arithmetics and the CV_32FC1 type for
messages. But it can also use fixed-point arithmetics and the CV_16SC1 message type for better
performance. To avoid an overflow in this case, the parameters must satisfy the following
requirement:

\f[10    \cdot 2^{levels-1}    \cdot max \_ data \_ term < SHRT \_ MAX\f]

@sa StereoMatcher
 */
```

```
class CV_EXPORTS_W StereoBeliefPropagation : public cv::StereoMatcher
{
public:
    using cv::StereoMatcher::compute;

    /** @overload */
    CV_WRAP virtual void compute(InputArray left, InputArray right, OutputArray disparity, Stream& stream) = 0;

    /** @brief Enables the stereo correspondence operator that finds the disparity for the specified data cost.

    @param data User-specified data cost, a matrix of msg_type type and
    Size(\<image columns\>\*ndisp, \<image rows\>) size.
    @param disparity Output disparity map. If disparity is empty, the output type is CV_16SC1 .
    Otherwise, the type is retained. In 16-bit signed format, the disparity values do not have
    fractional bits.
    @param stream Stream for the asynchronous version.
     */
    CV_WRAP virtual void compute(InputArray data, OutputArray disparity, Stream& stream = Stream::Null()) = 0;

    //! number of BP iterations on each level
    CV_WRAP virtual int getNumIters() const = 0;
    CV_WRAP virtual void setNumIters(int iters) = 0;

    //! number of levels
    CV_WRAP virtual int getNumLevels() const = 0;
    CV_WRAP virtual void setNumLevels(int levels) = 0;

    //! truncation of data cost
    CV_WRAP virtual double getMaxDataTerm() const = 0;
    CV_WRAP virtual void setMaxDataTerm(double max_data_term) = 0;

    //! data weight
    CV_WRAP virtual double getDataWeight() const = 0;
    CV_WRAP virtual void setDataWeight(double data_weight) = 0;

    //! truncation of discontinuity cost
    CV_WRAP virtual double getMaxDiscTerm() const = 0;
    CV_WRAP virtual void setMaxDiscTerm(double max_disc_term) = 0;
```

```
    //! discontinuity single jump
    CV_WRAP virtual double getDiscSingleJump() const = 0;
    CV_WRAP virtual void setDiscSingleJump(double disc_single_jump) = 0;

    //! type for messages (CV_16SC1 or CV_32FC1)
    CV_WRAP virtual int getMsgType() const = 0;
    CV_WRAP virtual void setMsgType(int msg_type) = 0;

    /** @brief Uses a heuristic method to compute the recommended parameters ( ndisp, iters and levels ) for the
    specified image size ( width and height ).
     */
    CV_WRAP static void estimateRecommendedParams(int width, int height, int& ndisp, int& iters, int& levels);
};

/** @brief Creates StereoBeliefPropagation object.

@param ndisp Number of disparities.
@param iters Number of BP iterations on each level.
@param levels Number of levels.
@param msg_type Type for messages. CV_16SC1 and CV_32FC1 types are supported.
 */
CV_EXPORTS_W Ptr<cuda::StereoBeliefPropagation>
    createStereoBeliefPropagation(int ndisp = 64, int iters = 5, int levels = 5, int msg_type = CV_32F);

//////////////////////////////////////////
// StereoConstantSpaceBP

/** @brief Class computing stereo correspondence using the constant space belief propagation algorithm. :

The class implements algorithm described in @cite Yang2010 . StereoConstantSpaceBP supports both local
minimum and global minimum data cost initialization algorithms. For more details, see the paper mentioned above. By default, a local algorithm is used. To enable a global algorithm, set use_local_init_data_cost to false .

StereoConstantSpaceBP uses a truncated linear model for the data cost and discontinuity terms:
```

\f[DataCost = data _ weight \cdot \min (\lvert I_2-I_1 \rvert , max _ data _ term)\f]

\f[DiscTerm = \min (disc _ single _ jump \cdot \lvert f_1-f_2 \rvert , max _ disc _ term)\f]

For more details, see @cite Yang2010 .

By default, StereoConstantSpaceBP uses floating-point arithmetics and the CV_32FC1 type for messages. But it can also use fixed-point arithmetics and the CV_16SC1 message type for better performance. To avoid an overflow in this case, the parameters must satisfy the following requirement:

\f[10 \cdot 2^{levels-1} \cdot max _ data _ term < SHRT _ MAX\f]

*/
class CV_EXPORTS_W StereoConstantSpaceBP : public cuda::StereoBeliefPropagation
{
public:
//! number of active disparity on the first level
CV_WRAP virtual int getNrPlane() const = 0;
CV_WRAP virtual void setNrPlane(int nr_plane) = 0;

CV_WRAP virtual bool getUseLocalInitDataCost() const = 0;
CV_WRAP virtual void setUseLocalInitDataCost(bool use_local_init_data_cost) = 0;

/** @brief Uses a heuristic method to compute parameters (ndisp, iters, levelsand nrplane) for the specified
image size (widthand height).
*/
CV_WRAP static void estimateRecommendedParams(int width, int height, int& ndisp, int& iters, int& levels, int& nr_plane);
};

/** @brief Creates StereoConstantSpaceBP object.

@param ndisp Number of disparities.
@param iters Number of BP iterations on each level.
@param levels Number of levels.
@param nr_plane Number of disparity levels on the first level.
@param msg_type Type for messages. CV_16SC1 and CV_32FC1 types are supported.
*/
CV_EXPORTS_W Ptr<cuda::StereoConstantSpaceBP>

createStereoConstantSpaceBP(int ndisp = 128, int iters = 8, int levels = 4, int nr_plane = 4, int msg_type = CV_32F);

//
// StereoSGM

/** @brief The class implements the modified H. Hirschmuller algorithm @cite HH08.
Limitation and difference are as follows:

- By default, the algorithm uses only 4 directions which are horizontal and vertical path instead of 8.
Set mode=StereoSGM::MODE_HH in createStereoSGM to run the full variant of the algorithm.
- Mutual Information cost function is not implemented.
Instead, Center-Symmetric Census Transform with \f$9 \times 7\f$ window size from @cite Spangenberg2013
is used for robustness.

@sa cv::StereoSGBM
*/
class CV_EXPORTS_W StereoSGM : public cv::StereoSGBM
{
public:
/** @brief Computes disparity map for the specified stereo pair

@param left Left 8-bit or 16-bit unsigned single-channel image.
@param right Right image of the same size and the same type as the left one.
@param disparity Output disparity map. It has the same size as the input images.
StereoSGM computes 16-bit fixed-point disparity map (where each disparity value has 4 fractional bits).
*/
CV_WRAP virtual void compute(InputArray left, InputArray right, OutputArray disparity) CV_OVERRIDE = 0;

/** @brief Computes disparity map with specified CUDA Stream

@sa compute
*/
CV_WRAP_AS(compute_with_stream) virtual void compute(InputArray left, InputArray right, OutputArray disparity, Stream& stream) = 0;
};

/** @brief Creates StereoSGM object.

@param minDisparity Minimum possible disparity value. Normally, it is zero but sometimes rectification algorithms can shift images, so this parameter needs to be adjusted accordingly.
@param numDisparities Maximum disparity minus minimum disparity. The value must be 64, 128 or 256.
@param P1 The first parameter controlling the disparity smoothness.This parameter is used for the case of slanted surfaces (not fronto parallel).
@param P2 The second parameter controlling the disparity smoothness.This parameter is used for "solving" the depth discontinuities problem.
@param uniquenessRatio Margin in percentage by which the best (minimum) computed cost function
value should "win" the second best value to consider the found match correct. Normally, a value within the 5-15 range is good enough.
@param mode Set it to StereoSGM::MODE_HH to run the full-scale two-pass dynamic programming algorithm.
It will consume O(W*H*numDisparities) bytes. By default, it is set to StereoSGM::MODE_HH4.
*/
CV_EXPORTS_W Ptr<cuda::StereoSGM> createStereoSGM(int minDisparity = 0, int numDisparities = 128, int P1 = 10, int P2 = 120, int uniquenessRatio = 5, int mode = cv::cuda::StereoSGM::MODE_HH4);

//
// DisparityBilateralFilter

/** @brief Class refining a disparity map using joint bilateral filtering. :

The class implements @cite Yang2010 algorithm.
*/
class CV_EXPORTS_W DisparityBilateralFilter : public cv::Algorithm
{
public:
    /** @brief Refines a disparity map using joint bilateral filtering.

    @param disparity Input disparity map. CV_8UC1 and CV_16SC1 types are supported.
    @param image Input image. CV_8UC1 and CV_8UC3 types are supported.
    @param dst Destination disparity map. It has the same size and type as disparity .
    @param stream Stream for the asynchronous version.
    */
    CV_WRAP virtual void apply(InputArray disparity, InputArray image, OutputArray dst, Stream& stream = Stream::Null()) = 0;

```
        CV_WRAP virtual int getNumDisparities() const = 0;
        CV_WRAP virtual void setNumDisparities(int numDisparities) = 0;

        CV_WRAP virtual int getRadius() const = 0;
        CV_WRAP virtual void setRadius(int radius) = 0;

        CV_WRAP virtual int getNumIters() const = 0;
        CV_WRAP virtual void setNumIters(int iters) = 0;

        //! truncation of data continuity
        CV_WRAP virtual double getEdgeThreshold() const = 0;
        CV_WRAP virtual void setEdgeThreshold(double edge_threshold) = 0;

        //! truncation of disparity continuity
        CV_WRAP virtual double getMaxDiscThreshold() const = 0;
        CV_WRAP virtual void setMaxDiscThreshold(double max_disc_threshold) = 0;

        //! filter range sigma
        CV_WRAP virtual double getSigmaRange() const = 0;
        CV_WRAP virtual void setSigmaRange(double sigma_range) = 0;
};

/** @brief Creates DisparityBilateralFilter object.

@param ndisp Number of disparities.
@param radius Filter radius.
@param iters Number of iterations.
 */
CV_EXPORTS_W Ptr<cuda::DisparityBilateralFilter>
        createDisparityBilateralFilter(int ndisp = 64, int radius = 3, int iters = 1);

//////////////////////////////////////////
// Utility

/** @brief Reprojects a disparity image to 3D space.

@param disp Input single-channel 8-bit unsigned, 16-bit signed, 32-bit signed or 32-bit
floating-point disparity image. If 16-bit signed format is used, the values are assumed to have no
fractional bits.
@param xyzw Output 3- or 4-channel floating-point image of the same size as disp . Each element of
xyzw(x,y) contains 3D coordinates (x,y,z) or (x,y,z,1) of the point (x,y) , computed from the
```

disparity map.
@param Q \f$4 \times 4\f$ perspective transformation matrix that can be obtained via stereoRectify .
@param dst_cn The number of channels for output image. Can be 3 or 4.
@param stream Stream for the asynchronous version.

@sa reprojectImageTo3D
*/
CV_EXPORTS void reprojectImageTo3D(InputArray disp, OutputArray xyzw, InputArray Q, int dst_cn = 4, Stream& stream = Stream::Null());
CV_EXPORTS_W inline void reprojectImageTo3D(GpuMat disp, CV_OUT GpuMat& xyzw, Mat Q, int dst_cn = 4, Stream& stream = Stream::Null()) {
    reprojectImageTo3D((InputArray)disp, (OutputArray)xyzw, (InputArray)Q, dst_cn, stream);
}

/** @brief Colors a disparity image.

@param src_disp Input single-channel 8-bit unsigned, 16-bit signed, 32-bit signed or 32-bit floating-point disparity image. If 16-bit signed format is used, the values are assumed to have no fractional bits.
@param dst_disp Output disparity image. It has the same size as src_disp. The type is CV_8UC4 in BGRA format (alpha = 255).
@param ndisp Number of disparities.
@param stream Stream for the asynchronous version.

This function draws a colored disparity map by converting disparity values from [0..ndisp) interval first to HSV color space (where different disparity values correspond to different hues) and then converting the pixels to RGB for visualization.
*/
CV_EXPORTS_W void drawColorDisp(InputArray src_disp, OutputArray dst_disp, int ndisp, Stream& stream = Stream::Null());

//! @}

}} // namespace cv { namespace cuda {